



University of Warsaw

# UW 2137

Simon Lindholm, Johan Sannemo, Mårten Wiman

2024-09-20

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

Contest (1)

sol.cpp28 lines

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

#define rep(i, a, b) for (int i = (a); i < (b); i++)
#define all(x) begin(x), end(x)
#define sz(x) int((x).size())
using ll = long long;
using pii = pair<int, int>;
using vi = vector<int>;

#ifdef LOCAL
auto& operator<<(auto&, pair<auto, auto>);
auto operator<<(auto& o, auto x) -> decltype(x.end(), o) {
    o << "{";
    for (int i = 0; auto y : x) o << ", " + !i++ * 2 << y;
    return o << "}";
}
auto& operator<<(auto& o, pair<auto, auto> x) {
    return o << "(" + x.first << ", " << x.second << ")";
}
void __print(auto... x) { ((cerr << " " << x), ...) << endl; }
#define debug(x...) cerr << "[" #x "]:", __print(x)
#else
#define debug(...) 2137
#endif

int main() {
    cin.tie(0)->sync_with_stdio(0);
}
```

.vimrc8 lines

```
set nu et ts=2 sw=2
filetype indent on
syntax on
colorscheme habamax
hi MatchParen ctermfg=66 ctermbg=234 cterm=underline
```

1

1

2

3

7

9

10

15

20

21

```
nnoremap ; :
nnoremap ; ;
inoremap {<cr> {<cr><esc>O <bs>

.bashrc3 lines
alias rm='trash'
alias mv='mv -i'
alias cp='cp -i'

Makefile6 lines
CXXFLAGS=-std=c++20 -Wall -Wextra -Wshadow
sol: sol.cpp
    g++ $(CXXFLAGS) -fsanitize=address,undefined -g -DLOCAL \
        sol.cpp -o sol
fast: sol.cpp
    g++ $(CXXFLAGS) -O2 sol.cpp -o fast

hash.sh3 lines
# 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

test.sh6 lines
#!/bin/bash
for ((i=1;i>0;i++)) do
    echo "$i"
    echo "$i" | ./gen > int
    diff -w <(. /sol < int) <(. /slow < int) || break
done
```

Mathematics (2)

2.1 Trigonometry

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

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

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

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

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

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

2.2 Geometry

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

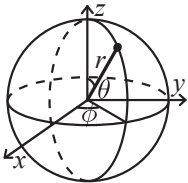
2.2.2 Quadrilaterals

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

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

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

2.2.3 Spherical coordinates



$$x = r \sin \theta \cos \phi$$
$$y = r \sin \theta \sin \phi$$
$$z = r \cos \theta$$

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = \operatorname{acos}(z/\sqrt{x^2 + y^2 + z^2})$$
$$\phi = \operatorname{atan2}(y, x)$$

## 2.3 Derivatives/Integrals

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

Integration by parts:

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

## 2.4 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$
$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

## 2.5 Series

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

## Data structures (3)

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change null\_type.

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

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

using namespace \_\_gnu\_pbds;

template<class T>

using Tree = tree<T, null\_type, less<T>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

void example() {

Tree<int> t, t2; t.insert(8);

auto it = t.insert(10).first;

assert(it == t.lower\_bound(9));

assert(t.order\_of\_key(10) == 1);

assert(t.order\_of\_key(11) == 2);

assert(\*t.find\_by\_order(0) == 8);

t.join(t2); // assuming T < T2 or T > T2, merge t2 into t

}

HashMap.h

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

1489f7, 8 lines

#include <ext/pb\_ds/assoc\_container.hpp>

using namespace \_\_gnu\_pbds;

// To use most bits rather than just the lowest ones:

struct chash { // large odd number for C

const uint64\_t C = 11(4e18 \* acos(0)) | 71;

ll operator()(ll x) const { return \_\_builtin\_bswap64(x\*C); }

};

gp\_hash\_table<ll, ll, chash> h({},{},{},{},{1<<16});

LazySegtree.h

Description: Basic segment tree template with lazy propagation. Can be easily extended with advanced functionality.

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

struct ST {

struct Node {};

int n;

vector<Node> t;

ST(int \_n) : n(\_n) { t.resize(2 \* n); }

Node join(const Node& a, const Node& b) {

void push(int u, int len) {} // push to u + 1, u + (len & -2)

void rec(int u,int lo,int hi,int l,int r,bool mod,auto f) {

if (l <= lo && hi <= r) return f(u, lo, hi);

push(u, hi - lo);

int mid = (lo + hi) / 2;

if (mid > l) rec(u + 1, lo, mid, l, r, mod, f);

if (mid < r) rec(u + (mid-lo) \* 2, mid, hi, l, r, mod, f);

if (mod) t[u] = join(t[u + 1], t[u + (mid - lo) \* 2]);

}

Node get(int l, int r) {

bool in = 0;

Node res;

rec(0, 0, n, l, r, 0, [&](int u, int, int) {

res = in ? join(res, t[u]) : t[u], in = 1;

});

return res;

}

void modify(int l, int r) {

rec(0, 0, n, l, r, 1, [&](int u, int lo, int hi) {});

}

};

UnionFindRollback.h

Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().

Usage: int t = uf.time(); ...; uf.rollback(t);

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

de4ad0, 21 lines

struct RollbackUF {

vi e; vector<pii> st;

RollbackUF(int n) : e(n, -1) {}

int size(int x) { return -e[find(x)]; }

int find(int x) { return e[x] < 0 ? x : find(e[x]); }

int time() { return sz(st); }

void rollback(int t) {

for (int i = time(); i --> t;)

e[st[i].first] = st[i].second;

st.resize(t);

}

bool join(int a, int b) {

a = find(a), b = find(b);

if (a == b) return false;

if (e[a] > e[b]) swap(a, b);

st.push\_back({a, e[a]});

st.push\_back({b, e[b]});

e[a] += e[b]; e[b] = a;

return true;

}

};

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming (“convex hull trick”).

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

8ec1c7, 30 lines

struct Line {

mutable ll k, m, p;

bool operator<(const Line& o) const { return k < o.k; }

bool operator<(ll x) const { return p < x; }

};

struct LineContainer : multiset<Line, less<>> {

// (for doubles, use inf = 1/.0, div(a,b) = a/b)

static const ll inf = LLONG\_MAX;

ll div(ll a, ll b) { // floored division

return a / b - ((a ^ b) < 0 && a % b); }

bool isect(iterator x, iterator y) {

if (y == end()) return x->p = inf, 0;

if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;

else x->p = div(y->m - x->m, x->k - y->k);

return x->p >= y->p;

}

void add(ll k, ll m) {

auto z = insert({k, m, 0}), y = z++, x = y;

while (isect(y, z)) z = erase(z);

if (x != begin() && isect(--x, y)) isect(x, y = erase(y));

while ((y = x) != begin() && (--x)->p >= y->p)

isect(x, erase(y));

}

ll query(ll x) {

assert(!empty());

auto l = \*lower\_bound(x);

return l.k \* x + l.m;

}

};

Treap.h
**Description:** Treap with lazy propagation and parent information. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
**Time:**  $\mathcal{O}(\log n)$

```
mt19937 rng(2137);
struct Node {
    Node *l = 0, *r = 0, *p = 0;
    int val, pr, c = 1;
    Node(int x) : val(x), pr(rng()) {}
    void pull();
    void push();
};
```

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

```
pair<Node*, Node*> split(Node* n, int k) {
    if (!n) return {};
    n->push();
    if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
        auto pa = split(n->l, k);
        n->l = pa.second;
        n->p = 0;
        if (n->l) n->l->p = n;
        n->pull();
        return {pa.first, n};
    } else {
        auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
        n->r = pa.first;
        n->p = 0;
        if (n->r) n->r->p = n;
        n->pull();
        return {n, pa.second};
    }
}
Node* merge(Node* l, Node* r) {
    if (!l || !r) return l ? r;
    if (l->pr > r->pr) {
        l->push();
        l->r = merge(l->r, r);
        l->r->p = l;
        l->pull();
        return l;
    } else {
        r->push();
        r->l = merge(l, r->l);
        r->l->p = r;
        r->pull();
        return r;
    }
}
```

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

```
struct FT {
    vector<ll> s;
    FT(int n) : s(n) {}
    void update(int pos, ll dif) { // a[pos] += dif
        for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
    }
    ll query(int pos) { // sum of values in [0, pos)
        ll res = 0;
        for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    }
};
```

```
return res;
}
int lower_bound(ll sum) { // min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >= 1) {
        if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
            pos += pw, sum -= s[pos-1];
    }
    return pos;
}
};
```

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

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

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

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

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

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

```
vi mo(vector<pii> Q) {
    int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
    vi s(sz(Q)), res = s;
    #define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
    iota(all(s), 0);
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
    for (int qi : s) {
        pii q = Q[qi];
        while (L > q.first) add(--L, 0);
        while (R < q.second) add(R++, 1);
        while (L < q.first) del(L++, 0);
        while (R > q.second) del(--R, 1);
        res[qi] = calc();
    }
    return res;
}
```

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

## Numerical (4)

### 4.1 Polynomials and recurrences

Polynomial.h
**struct** Poly {
 vector<double> a;
 double operator()(double x) const {

```
double val = 0;
for (int i = sz(a); i--;) (val *= x) += a[i];
return val;
}

void diff() {
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
}

void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
}
};
```

**PolyRoots.h**  
**Description:** Finds the real roots to a polynomial.  
**Usage:** polyRoots({{2,-3,1}},-1e9,1e9) // solve x<sup>2</sup>-3x+2 = 0  
**Time:**  $\mathcal{O}(n^2 \log(1/\epsilon))$

```
"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:  $p(x) = a[0] * x^0 + \dots + a[n-1] * x^{n-1}$ . For numerical precision, pick  $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \dots n-1$ .  
**Time:**  $\mathcal{O}(n^2)$

```
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;
}
```

**BerlekampMassey.h**  
**Description:** Recovers any  $n$ -order linear recurrence relation from the first  $2n$  terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for

```
floats is not guaranteed. Output will have size  $\leq n$ .
Usage: berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}
Time:  $\mathcal{O}(N^2)$ 

"/number-theory/ModInt.h" s10031, 20 lines

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

    mint b = 1;
    rep(i,0,n) { ++m;
        mint d = s[i];
        rep(j,1,L+1) d += C[j] * s[i - j];
        if (d == 0) continue;
        T = C; mint coef = d / b;
        rep(j,m,n) C[j] -= coef * B[j - m];
        if (2 * L > i) continue;
        L = i + 1 - L; B = T; b = d; m = 0;
    }

    C.resize(L + 1); C.erase(C.begin());
    for (mint& x : C) x *= -1;
    return C;
}
```

**LinearRecurrence.h**  
**Description:** Generates the  $k$ 'th term of an  $n$ -order linear recurrence  $S[i] = \sum_j S[i - j - 1]tr[j]$ , given  $S[0 \dots \geq n - 1]$  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 and no local maximum. The maximum error in the result is  $\epsilon$ ps. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

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

31d45b, 14 lines

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  
Seeef, 14 lines

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.

```
4756fc, 7 lines

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

aa8530, 68 lines

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

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

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

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

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

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

```
T solve(vd &x) {
```

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

4.3 Matrices

Determinant.h

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

bd5cec, 15 lines

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

IntDeterminant.h

**Description:** Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.

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

3313dc, 18 lines

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step
                ll t = a[i][i] / a[j][i];
                if (t) rep(k,i,n)
                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1;
            }
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    }
    return (ans + mod) % mod;
}
```

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

44c9ab, 38 lines

```
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" fa2d7a, 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  $F_2$ . If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys  $A$  and  $b$ .  
**Time:**  $\mathcal{O}(n^2 m)$

fa2d7a, 34 lines

```
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0);
    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)
}
```

MatrixInverse.h

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

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

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

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

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

Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

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

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

BlackBoxDet.h

**Description:** Black-box algorithm for the determinant of a matrix.  $f$  should be a function s.t.  $f(v) = Mv$ . Can add multiple iterations if order of recurrence is less than  $n$ .  
**Time:**  $\mathcal{O}(n^2 + 2n \text{ calls to } f)$

```
"BerlekampMassey.h"
mt19937_64 rng(2137);
mint det(int n, auto f) {
    auto rnd = [&]() {
        vector<mint> v(n);
        rep(i, 0, n) v[i] = rng() % mint::MOD;
        return v;
    };
    auto a = rnd(), b = rnd(), c = rnd();
    vector<mint> s(2 * n);
    rep(i, 0, 2 * n) {
        rep(j, 0, n) s[i] += a[j] * b[j];
        rep(j, 0, n) b[j] *= c[j];
        b = f(move(b));
    }
}
```

```
}
auto v = berlekampMassey(s);
if (sz(v) != n) return 0;
mint p = 1;
rep(i, 0, n) p *= c[i];
return v[n-1] / p * (n % 2 ? 1 : -1);
}
```

4.4 Fourier transforms

FFT.h

**Description:** Multiply polynomials for any modulus. Works for  $n+m \leq 2^{24}$  and  $c_k \leq 5 \cdot 10^{25}$ .  
**Time:**  $\mathcal{O}((n+m) \log(n+m))$

```
"../number-theory/ModInt.h"
template<class T>
void ntt(vector<T>& a, bool inv) {
    int n = sz(a); vector<T> b(n);
    for (int i = n / 2; i; i /= 2, swap(a, b)) {
        T w = T(T::ROOT).pow((T::MOD - 1) / n * i), m = 1;
        for (int j = 0; j < n; j += 2 * i, m *= w) rep(k, 0, i) {
            T u = a[j+k], v = a[j+k+i] * m;
            b[j/2+k] = u+v, b[j/2+k+n/2] = u-v;
        }
    }
    if (inv) {
        reverse(1 + all(a));
        T z = T(n).inv(); rep(i, 0, n) a[i] *= z;
    }
}

template<class T>
vector<T> conv(vector<T> a, vector<T> b) {
    int s = sz(a) + sz(b) - 1, n = 1 << __lg(2 * s - 1);
    a.resize(n); ntt(a, 0); b.resize(n); ntt(b, 0);
    rep(i, 0, n) a[i] *= b[i];
    ntt(a, 1); a.resize(s);
    return a;
}

template<class T>
vector<T> mconv(const auto& x, const auto& y) {
    auto con = [&](const auto& v) {
        vector<T> w(sz(v)); rep(i, 0, sz(v)) w[i] = v[i].x;
        return w;
    };
    return conv(con(x), con(y));
}

template<class T>
vector<T> conv3(const vector<T>& a, const vector<T>& b) {
    using m0 = Mod<754974721, 11>; auto c0 = mconv<m0>(a, b);
    using m1 = Mod<167772161, 3>; auto c1 = mconv<m1>(a, b);
    using m2 = Mod<469762049, 3>; auto c2 = mconv<m2>(a, b);
    int n = sz(c0); vector<T> d(n); m1 r01 = m1(m0::MOD).inv();
    m2 r02 = m2(m0::MOD).inv(), r12 = m2(m1::MOD).inv();
    rep(i, 0, n) {
        int x = c0[i].x, y = ((c1[i] - x) * r01).x,
            z = (((c2[i] - x) * r02 - y) * r12).x;
        d[i] = (T(z) * m1::MOD + y) * m0::MOD + x;
    }
    return d;
}
```

FastSubsetTransform.h

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

```
void FST(vi& a, bool inv) {
```

```
    for (int n = sz(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
            int &u = a[j], &v = a[j + step]; tie(u, v) =
                inv ? pii(v - u, u) : pii(v, u + v); // AND
                inv ? pii(v, u - v) : pii(u + v, u); // OR
                pii(u + v, u - v); // XOR
        }
    }
    if (inv) for (int& x : a) x /= sz(a); // XOR only
}
vi conv(vi a, vi b) {
    FST(a, 0); FST(b, 0);
    rep(i,0,sz(a)) a[i] *= b[i];
    FST(a, 1); return a;
}
```

## Number theory (5)

### 5.1 Modular arithmetic

#### ModInt.h

**Description:** Operators for modular arithmetic.

a902ca, 28 lines

```
template<int M, int R>
struct Mod {
    static const int MOD = M, ROOT = R;
    int x;
    Mod(ll y = 0) : x(y % M) { x += (x < 0) * M; }
    Mod& operator+=(Mod o) {
        if ((x += o.x) >= M) x -= M;
        return *this; }
    Mod& operator--(Mod o) {
        if ((x -= o.x) < 0) x += M;
        return *this; }
    Mod& operator*=(Mod o) {
        x = 1ll * x * o.x % M;
        return *this; }
    Mod& operator/=(Mod o) { return *this *= o.inv(); }
    friend Mod operator+(Mod a, Mod b) { return a += b; }
    friend Mod operator-(Mod a, Mod b) { return a -= b; }
    friend Mod operator*(Mod a, Mod b) { return a *= b; }
    friend Mod operator/(Mod a, Mod b) { return a /= b; }
    auto operator<=>(const Mod&) const = default;
    Mod pow(ll n) const {
        Mod a = x, b = 1;
        for (; n; n /= 2, a *= a) if (n & 1) b *= a;
        return b;
    }
    Mod inv() const { assert(x != 0); return pow(M - 2); }
};
using mint = Mod<998244353, 3>;
```

#### ModInverse.h

**Description:** Pre-computation of modular inverses. Assumes  $\text{LIM} \leq \text{mod}$  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;
```

#### ModPow.h

b83e45, 8 lines

```
const ll mod = 1000000007; // faster if const
```

```
ll modpow(ll b, ll e) {
    ll ans = 1;
```

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

#### ModLog.h

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

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

c040b8, 11 lines

```
ll modLog(ll a, ll b, ll m) {
    ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
    unordered_map<ll, ll> A;
    while (j <= n && (e = f = e * a % m) != b % m)
        A[e * b % m] = j++;
    if (e == b % m) return j;
    if (__gcd(m, e) == __gcd(m, b))
        rep(i,2,n+2) if (A.count(e = e * f % m))
            return n * i - A[e];
    return -1;
}
```

#### ModSum.h

**Description:** Sums of mod'ed arithmetic progressions.

$\text{modsum}(\text{to}, c, k, m) = \sum_{i=0}^{\text{to}-1} (ki + c) \% m$ . `divsum` is similar but for floored division.

**Time:**  $\log(m)$ , with a large constant.

9c796e, 15 lines

```
ll sumsq(ll to) { return to / 2 * ((to-1) | 1); }

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

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

#### ModMulLL.h

**Description:** Calculate  $a \cdot b \bmod c$  (or  $a^b \bmod c$ ).

**Time:**  $\mathcal{O}(1)$  for `modmul`,  $\mathcal{O}(\log b)$  for `modpow`

02ea06, 9 lines

```
ll modmul(ll a, ll b, ll M) {
    return (__int128)a * b % M;
}
ll modpow(ll b, ll e, ll mod) {
    ll ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans;
}
```

#### ModSqrt.h

**Description:** Tonelli-Shanks algorithm for modular square roots. Finds  $x$  s.t.  $x^2 = a \pmod p$  ( $-x$  gives the other solution).

**Time:**  $\mathcal{O}(\log^2 p)$  worst case,  $\mathcal{O}(\log p)$  for most  $p$

b7cab4, 24 lines

```
ll sqrt(ll a, ll p) {
    a %= p; if (a < 0) a += p;
    if (a == 0) return 0;
    if (modpow(a, (p-1)/2, p) != 1) return -1; // 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;
    }
}
```

#### ModGen.h

**Description:** Finds a primitive root modulo  $p$ .

ff3110, 8 lines

```
mt19937_64 rng(2137);
ll modGen(ll n) {
    map<ll, int> f; factor(n - 1, f); rep:
    ll g = rng() % (n - 1) + 1;
    for (auto [p, _] : f)
        if (modpow(g, (n - 1) / p, n) == 1) goto rep;
    return g;
}
```

#### ModArith.h

**Description:** Statistics on a mod'ed arithmetic sequence.

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

45f202, 32 lines

```
"Euclid.h"
ll cdiv(ll x, ll y) { return x / y + ((x ^ y) > 0 && x % y); }
```

```
// min (ax + b) % m for 0 <= x <= n
ll minRemainder(ll a, ll b, ll m, ll n) {
    assert(a >= 0 && m > 0 && b >= 0 && n >= 0);
    a %= m, b %= m; n = min(n, m - 1);
    if (a == 0) return b;
    if (b >= a) {
        ll ad = cdiv(m - b, a);
        n -= ad; if (n < 0) return b;
        b += ad * a - m;
    }
    ll q = m / a, m2 = m % a;
    if (m2 == 0) return b;
    if (b / m2 > n / q) return b - n / q * m2;
    n -= b / m2 * q; b %= m2;
    ll y2 = (n * a + b) / m;
    ll x2 = cdiv(m2 * y2 - b, a);
    if (x2 * a - m2 * y2 + b >= m2) --x2;
    return minRemainder(a, b, m2, x2);
}
```

```
// min x >= 0 s.t. l <= (ax + b) % m <= r
ll minBetween(ll a, ll b, ll m, ll l, ll r) {
    ll x, y, g = euclid(a, m, x, y);
    if (g > 1)
        return minBetween(a/g,b/g,m/g,l/g+(l%g>b%g),r/g-(r%g<b%g));
    if (l > r) return -1; // no solution
    if ((x % m) < 0) x += m;
    ll b2 = (l - b) * x % m;
    return minRemainder(x, b2 < 0 ? b2 + m : b2, m, r - 1);
}
```



## 5.2 Primality

### FastEratosthenes.h

**Description:** Prime sieve for generating all primes smaller than LIM.

**Time:** LIM=1e9  $\approx$  1.5s 6b2912, 20 lines

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

### MillerRabin.h

**Description:** Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to  $7 \cdot 10^{18}$ ; for larger numbers, use Python and extend A randomly.

**Time:** 7 times the complexity of  $a^b \bmod c$ .

```
"ModMulLL.h"
418a8d, 12 lines

bool isPrime(ll n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ll A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ll a : A) { // ^ count trailing zeroes
        ll 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;
}
```

### Factor.h

**Description:** Pollard-rho randomized factorization algorithm.

**Time:**  $\mathcal{O}\left(n^{1/4}\right)$ , less for numbers with small factors.

```
"ModMulLL.h", "MillerRabin.h"
0750d1, 16 lines

ll pollard(ll n) {
    ll x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    auto f = [&](ll k) { return modmul(k, k, n) + i; };
    while (t++ % 40 || __gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}

void factor(ll n, map<ll, int>& cnt) {
    if (n == 1) return;
    if (isPrime(n)) { cnt[n]++; return; }
    ll x = pollard(n);
    factor(x, cnt); factor(n / x, cnt);
}
```

## 5.3 Divisibility

### Euclid.h

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

```
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  $x \equiv a \pmod m$ ,  $x \equiv b \pmod n$ . If  $|a| < m$  and  $|b| < n$ ,  $x$  will obey  $0 \leq x < \text{lcm}(m, n)$ . Assumes  $mn < 2^{62}$ .

```
Time: log(n)
"Euclid.h"
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;
}
```

### SameDiv.h

**Description:** Divides the interval  $[1, \infty)$  into constant division intervals.

```
Time:  $\mathcal{O}(\sqrt{n})$ 
24617c, 13 lines

vector<ll> sameFloor(ll n) {
    vector<ll> v;
    for (ll i = 1; i <= n; i = n / (n / i) + 1) v.push_back(i);
    return v.push_back(n + 1), v;
}

vector<ll> sameCeil(ll n) {
    vector<ll> v;
    for (ll i = 1, j; i < n; i = (n + j - 2) / (j - 1)) {
        j = (n + i - 1) / i;
        v.push_back(i);
    }
    return v.push_back(n), v;
}
```

## 5.4 Fractions

### FracBinarySearch.h

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

**Usage:** `fracBS([](Frac f) { return f.p>=3*f.q; }, 10);` // {1,3}

```
Time:  $\mathcal{O}(\log(N))$ 
27ab3e, 25 lines

struct Frac { ll p, q; };

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

```
        if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
            adv -= step; si = 2;
        }
    }
    hi.p += lo.p * adv;
    hi.q += lo.q * adv;
    dir = !dir;
    swap(lo, hi);
    A = B; B = !!adv;
}
return dir ? hi : lo;
}
```

## 5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

## 5.6 Primes

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

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

## 5.7 Estimates

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

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  $n < 1e19$ .

## 5.8 Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

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

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

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

```
IntPerm.h
Description: Permutation -> integer conversion. (Not order preserving.)
Integer -> permutation can use a lookup table.
Time:  $\mathcal{O}(n)$ 
044568, 6 lines

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

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 Burnside’s lemma

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

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

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

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

<i>n</i>	0	1	2	3	4	5	6	7	8	9	20	50	100
<i>p</i> ( <i>n</i> )	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

6.2.2 Lucas’ Theorem

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

6.2.3 Binomials

```
multinomial.h
Description: Computes  $\binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \dots k_n!}$ .
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;
}
```

6.3 General purpose numbers

6.3.1 Bernoulli numbers

EGF of Bernoulli numbers is  $B(t) = \frac{t}{e^t - 1}$  (FFT-able).  $B[0, \dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

6.3.2 Stirling numbers of the first kind

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

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

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

6.3.3 Eulerian numbers

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

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

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

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

6.3.4 Stirling numbers of the second kind

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

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

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

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

6.3.5 Bell numbers

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

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

6.3.6 Labeled unrooted trees

# on  $n$  vertices:  $n^{n-2}$   
# on  $k$  existing trees of size  $n_i$ :  $n_1 n_2 \dots n_k n^{k-2}$   
# with degrees  $d_i$ :  $(n-2)! / ((d_1-1)! \dots (d_n-1)!)$

6.3.7 Catalan numbers

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)n!}$$
$$C_0 = 1, \ C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \ C_{n+1} = \sum C_i C_{n-i}$$
$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$$

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

Graph (7)

7.1 Fundamentals

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

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

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

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

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

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
    int n = sz(m);
```

```
    rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
    rep(k,0,n) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) {
            auto newDist = max(m[i][k] + m[k][j], -inf);
            m[i][j] = min(m[i][j], newDist);
        }
    rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

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

```
vi topoSort(const vector<vi>& gr) {
    vi indeg(sz(gr)), ret;
    for (auto& li : gr) for (int x : li) indeg[x]++;
    queue<int> q; // use priority_queue for lexic. largest ans.
    rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);
    while (!q.empty()) {
        int i = q.front(); // top() for priority queue
        ret.push_back(i);
        q.pop();
        for (int x : gr[i])
            if (--indeg[x] == 0) q.push(x);
    }
    return ret;
}
```

7.2 Network flow

**Dinic.h**  
**Description:** Flow algorithm with complexity  $\mathcal{O}(VE \log U)$  where  $U = \max|\text{cap}|$ .  $\mathcal{O}(\min(E^{1/2}, V^{2/3})E)$  if  $U = 1$ ;  $\mathcal{O}(\sqrt{VE})$  for bipartite matching.

```
struct Dinic {
    struct Edge {
        int to, rev;
        ll c, oc;
        ll flow() { return max(oc - c, 0LL); } // if you need flows
    };
    vi lvl, ptr, q;
    vector<vector<Edge>> adj;
    Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
    void addEdge(int a, int b, ll c, ll rcap = 0) {
        adj[a].push_back({b, sz(adj[b]), c, c});
        adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
    }
    ll dfs(int v, int t, ll f) {
        if (v == t || !f) return f;
        for (int& i = ptr[v]; i < sz(adj[v]); i++) {
            Edge& e = adj[v][i];
            if (lvl[e.to] == lvl[v] + 1)
                if (ll p = dfs(e.to, t, min(f, e.c))) {
                    e.c -= p, adj[e.to][e.rev].c += p;
                    return p;
                }
        }
        return 0;
    }
    ll calc(int s, int t) {
        ll flow = 0; q[0] = s;
        rep(L,0,31) do { // 'int L=30' maybe faster for random data
            lvl = ptr = vi(sz(q));
```

```
            int qi = 0, qe = lvl[s] = 1;
            while (qi < qe && !lvl[t]) {
                int v = q[qi++];
                for (Edge e : adj[v])
                    if (!lvl[e.to] && e.c >> (30 - L))
                        q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
            }
            while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
        } while (lvl[t]);
        return flow;
    }
    bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

**MinCostMaxFlow.h**  
**Description:** Min-cost max-flow. If costs can be negative, call `setpi` before `maxflow`, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.  
**Time:**  $\mathcal{O}(FE \log(V))$  where  $F$  is max flow.  $\mathcal{O}(VE)$  for `setpi`. f8c012, 79 lines

```
#include <ext/pb_ds/priority_queue.hpp>

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

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

```
MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {}

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

```
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 });

    while (!q.empty()) {
        s = q.top().second; q.pop();
        seen[s] = 1; di = dist[s] + pi[s];
        for (edge& e : ed[s]) if (!seen[e.to]) {
            ll val = di - pi[e.to] + e.cost;
            if (e.cap - e.flow > 0 && val < dist[e.to]) {
                dist[e.to] = val;
                par[e.to] = &e;
                if (its[e.to] == q.end())
                    its[e.to] = q.push({ -dist[e.to], e.to });
                else
                    q.modify(its[e.to], { -dist[e.to], e.to });
            }
        }
    }
    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 (edge* x = par[t]; x; x = par[x->from])
            fl = min(fl, x->cap - x->flow);

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

```
// 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 (edge& e : ed[i]) if (e.cap)
                if ((v = pi[i] + e.cost) < pi[e.to])
                    pi[e.to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
}
};
```

MinCut.h

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

GlobalMinCut.h

**Description:** Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.

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

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

GomoryHu.h

**Description:** Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.

**Time:**  $\mathcal{O}(V)$  Flow Computations

```
"PushRelabel.h"
0418b3, 13 lines

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

7.3 Matching

Matching.h

**Description:** Fast biparite matching algorithm. Graph  $g$  should be a list of neighbors of the left partition. Returns the match for every left vertex.

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

```
vi match(int n, int m, vector<vi>& g) {
    vi l(n, -1), r(m, -1), q(n), d(n);
    auto dfs = [&](auto f, int u) -> bool {
        int t = exchange(d[u], 0) + 1;
        for (int v : g[u])
            if (r[v] == -1 || (d[r[v]] == t && f(f, r[v])))
                return l[u] = v, r[v] = u, 1;
        return 0;
    };
    for (int t = 0, f = 0;; t = f = 0, d.assign(n, 0)) {
        rep(i, 0, n) if (l[i] == -1) q[t++] = i, d[i] = 1;
        rep(i, 0, t) for (int v : g[q[i]]) {
            if (r[v] == -1) f = 1;
            else if (!d[r[v]]) d[r[v]] = d[q[i]] + 1, q[t++] = r[v];
        }
        if (!f) return l;
        rep(i, 0, n) if (l[i] == -1) dfs(dfs, i);
    }
}
```

MinimumVertexCover.h

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

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

```
"DFSMatching.h"
da4196, 20 lines

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

WeightedMatching.h

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

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

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

Blossom.h

**Description:** Matching for general graphs.

**Time:**  $\mathcal{O}(nm)$ , fast in practice

```
vi blossom(vector<vi>& g) {
    int n = sz(g), t = -1;
    vi m(n, -1), l(n), p(n), o(n), b(n, -1), q;
    auto lca = [&](int x, int y) {
        for (t++; ; swap(x, y)) {
            if (x == -1) continue;
            if (b[x] == t) return x;
            b[x] = t;
            x = (m[x] == -1 ? -1 : o[p[m[x]]]);
        }
    };
    auto blossom = [&](int v, int w, int a) {
        while (o[v] != a) {
            p[v] = w; w = m[v];
            if (l[w] == 1) l[w] = 0, q.push_back(w);
            o[v] = o[w] = a; v = p[w];
        }
    };
    auto augment = [&](int v) {
        while (v != -1) {
            int pv = p[v], nv = m[pv];
            m[v] = pv; m[pv] = v; v = nv;
        }
    };
}
```

```
};
auto bfs = [&](int r) {
    fill(all(l), -1); iota(all(o), 0); q.clear();
    l[r] = 0; q.push_back(r);
    rep(i, 0, sz(q)) {
        int v = q[i];
        for (auto x : g[v]) {
            if (l[x] == -1) {
                l[x] = 1; p[x] = v;
                if (m[x] == -1) return augment(x), 1;
                l[m[x]] = 0; q.push_back(m[x]);
            } else if (l[x] == 0 && o[v] != o[x]) {
                int a = lca(o[v], o[x]);
                blossom(x, v, a); blossom(v, x, a);
            }
        }
    }
    return 0;
};
// Time halves if you start with (any) maximal matching.
rep(i, 0, n) if (m[i] == -1) bfs(i);
return m;
}
```

7.4 DFS algorithms

**SCC.h**  
**Description:** Finds strongly connected components in a directed graph. If vertices *u, v* belong to the same component, we can reach *u* from *v* and vice versa.  
**Usage:** scc(graph, [&](vi& v) { ... }) visits all components in reverse topological order. comp[i] holds the component index of a node (a component only has edges to components with lower index). ncomps will contain the number of components.  
**Time:**  $\mathcal{O}(E + V)$

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

c6b7c7, 32 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, top = me;
    for (auto [y, e] : ed[at]) if (e != par) {
        if (num[y]) {
            top = min(top, num[y]);
            if (num[y] < me)
                st.push_back(e);
        } 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:** Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type (a||b)&&!(a||c)&&(d||b)&&... becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).  
**Usage:** TwoSat ts(number of boolean variables);  
ts.either(0, ~3); // Var 0 is true or var 3 is false  
ts.setValue(2); // Var 2 is true  
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true  
ts.solve(); // Returns true iff it is solvable  
ts.values[0..N-1] holds the assigned values to the vars  
**Time:**  $\mathcal{O}(N + E)$ , where N is the number of boolean variables, and E is the number of clauses.

5f9706, 56 lines

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

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

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

    void either(int f, int j) {
        f = max(2*f, -1-2*f);
        j = max(2*j, -1-2*j);
    }
}
```

```
gr[f].push_back(j^1);
gr[j].push_back(f^1);
}
void setValue(int x) { either(x, x); }

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

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

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

428fec, 16 lines

EulerWalk.h

**Description:** Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of pairs (node, edge) in the Eulerian path/cycle with src at the start, or empty list if no cycle/path exists.  
**Time:**  $\mathcal{O}(V + E)$

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

---

b0d5b1, 12 lines

f7c0bc, 49 lines

bfce85, 25 lines

0f62fb, 21 lines

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

```
typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.time));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp);
    int m = sz(li)-1;
    rep(i,0,m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    }
}
```

```

sort(all(li), cmp);
li.erase(unique(all(li)), li.end());
rep(i,0,sz(li)) rev[li[i]] = i;
vpi ret = {pii(0, li[0])};
rep(i,0,sz(li)-1) {
    int a = li[i], b = li[i+1];
    ret.emplace_back(rev[lca.lca(a, b)], b);
}
return ret;
}

```

## HLD.h

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

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

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

```

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

```

## LinkCutTree.h

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

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

0fb462, 90 lines

```

struct Node { // Splay tree. Root's pp contains tree's parent.
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this;
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
    void pushFlip() {
        if (!flip) return;
        flip = 0; swap(c[0], c[1]);
        if (c[0]) c[0]->flip ^= 1;
        if (c[1]) c[1]->flip ^= 1;
    }
    int up() { return p ? p->c[1] == this : -1; }
    void rot(int i, int b) {
        int h = i ^ b;
        Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
        if ((y->p = p)) p->c[up()] = y;
        c[i] = z->c[i ^ 1];
        if (b < 2) {
            x->c[h] = y->c[h ^ 1];
            y->c[h ^ 1] = x;
        }
        z->c[i ^ 1] = this;
        fix(); x->fix(); y->fix();
        if (p) p->fix();
        swap(pp, y->pp);
    }
    void splay() {
        for (pushFlip(); p; ) {
            if (p->p) p->p->pushFlip();
            p->pushFlip(); pushFlip();
            int c1 = up(), c2 = p->up();
            if (c2 == -1) p->rot(c1, 2);
            else p->p->rot(c2, c1 != c2);
        }
    }
    Node* first() {
        pushFlip();
        return c[0] ? c[0]->first() : (splay(), this);
    }
};

struct LinkCut {
    vector<Node> node;
    LinkCut(int N) : node(N) {}

    void link(int u, int v) { // add an edge (u, v)
        assert(!connected(u, v));
        makeRoot(&node[u]);
        node[u].pp = &node[v];
    }
    void cut(int u, int v) { // remove an edge (u, v)
        Node *x = &node[u], *top = &node[v];
        makeRoot(top); x->splay();
        assert(top == (x->pp ? x->c[0]));
        if (x->pp) x->pp = 0;
        else {
            x->c[0] = top->p = 0;
            x->fix();
        }
    }
};

```

```

}
bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
}
void makeRoot(Node* u) {
    access(u);
    u->splay();
    if (u->c[0]) {
        u->c[0]->p = 0;
        u->c[0]->flip ^= 1;
        u->c[0]->pp = u;
        u->c[0] = 0;
        u->fix();
    }
}
Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
        pp->splay(); u->pp = 0;
        if (pp->c[1]) {
            pp->c[1]->p = 0; pp->c[1]->pp = pp;
            pp->c[1] = u; pp->fix(); u = pp;
        }
    }
    return u;
}
};

```

## DirectedMST.h

**Description:** Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

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

../data-structures/UnionFindRollback.h 39e620, 60 lines

```

struct Edge { int a, b; ll w; };
struct Node {
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? b : a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node& a) { a->prop(); a = merge(a->l, a->r); }

```

```

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node(e));
    ll res = 0;
    vi seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, {-1, -1}), comp;
    deque<tuple<int, int, vector<Edge>>> cycs;
    rep(s, 0, n) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1, {}};
            Edge e = heap[u]->top();

```

```
    heap[u]->delta -= e.w, pop(heap[u]);
    Q[qi] = e, path[qi++] = u, seen[u] = s;
    res += e.w, u = uf.find(e.a);
    if (seen[u] == s) {
        Node* cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.join(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
    }
}
rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
}

for (auto& [u,t,comp] : cycs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
}
rep(i,0,n) par[i] = in[i].a;
return {res, par};
}
```

7.8 Advanced

**KthWalk.h**  
**Description:** Eppstein's algorithm for the  $k$ -th shortest walk in a directed graph with non-negative edge weights.  
**Memory:**  $\mathcal{O}((n+m)\log n+k)$   
**Time:**  $\mathcal{O}((n+m)\log n+k\log k)$

f7b9b0, 53 lines

```
struct KthWalk {
    using Edge = pair<int, ll>;
    struct Node { // persistent leftist heap node
        Node *l = 0, *r = 0;
        int s; Edge e;
        Node(Edge _e) : e(_e) {}
    };
    ll d0;
    priority_queue<pair<ll, Node*>> q;
    vector<Node*> h;
    KthWalk(vector<vector<Edge>>& g, int s, int t) {
        int n = sz(g); vector<vector<Edge>> r(n);
        rep(i, 0, n) for (auto [j,w] : g[i]) r[j].push_back({i,w});
        vector<ll> d(n, LLONG_MAX);
        vi ord, p(n, -1);
        priority_queue<pair<ll, int>> pq;
        pq.push({d[t] = 0, t});
        while (sz(pq)) {
            auto [dd, u] = pq.top(); pq.pop();
            if (d[u] != -dd) continue;
            ord.push_back(u);
            for (auto [v, w] : r[u]) if (d[u] + w < d[v])
                pq.push({-d[v] = d[u] + w, v}); p[v] = u;
        }
        if ((d0 = d[s]) == LLONG_MAX) return;
        h.resize(n);
        for (int u : ord) {
            int pp = p[u]; if (pp != -1) h[u] = h[pp];
            for (auto [v, w] : g[u]) if (d[v] != LLONG_MAX) {
                ll x = w + d[v] - d[u];
                if (x || v != pp) h[u] = merge(h[u], new Node({v, x}));
                else pp = -1;
            }
        }
    }
    q.push({0, new Node({s, 0})});
}
```

```
    }
    Node* merge(Node* a, Node* b) {
        if (!a || !b) return a ?: b;
        if (a->e.second > b->e.second) swap(a, b);
        Node* c = new Node(*a); c->r = merge(c->r, b);
        if (!c->l || c->l->s < c->r->s) swap(c->l, c->r);
        c->s = (c->r ? c->r->s : 0) + 1; return c;
    }
    ll next() { // -1 if no path
        if (!sz(q)) return -1;
        auto [d, a] = q.top(); q.pop();
        if (a->l) q.push({d - a->l->e.second + a->e.second, a->l});
        if (a->r) q.push({d - a->r->e.second + a->e.second, a->r});
        Node* t = h[a->e.first];
        if (t) q.push({d - t->e.second, t});
        return d0 - d;
    }
};
```

7.9 Math

7.9.1 Number of Spanning Trees

Create an  $N \times N$  matrix  $\text{mat}$ , and for each edge  $a \rightarrow b \in G$ , do  $\text{mat}[a][b]--$ ,  $\text{mat}[b][b]++$  (and  $\text{mat}[b][a]--$ ,  $\text{mat}[a][a]++$  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).

7.9.2 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  $k = 1 \dots n$ ,

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

7.9.3 Gale-Ryser theorem

A simple bipartite graph with degree sequences  $a_1 \geq \dots \geq a_n$  and  $b_1, \dots, b_m$  exists iff  $\sum a_i = \sum b_i$  and for every  $1 \leq k \leq n$

$$\sum_{i=1}^k a_i \leq \sum_{i=1}^m \min(b_i, k).$$

7.9.4 BEST theorem

The number of Eulerian circuits on an Eulerian graph equals

$$t(v) \prod_u (\deg(u) - 1)!$$

where  $t(v)$  is the number of spanning trees directed towards an arbitrary root  $v$ , and  $\deg(u)$  is the outdegree of vertex  $u$ .

Geometry (8)

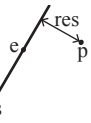
8.1 Geometric primitives

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

3e64f3, 26 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    auto operator<=>((const P&) const = default;
    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 << ")"; }
};
```

**LineDist.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.



f6bf6b, 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();
}
```

**SegDist.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;



5c88f4, 6 lines

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



```
Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
"PPoint.h", "OnSegment.h" 9d5f7f2_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)};
}
```

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

**Usage:** `auto res = lineInter(s1,e1,s2,e2);`  
`if (res.first == 1)`  
`cout << "intersection point at " << res.second << endl;`  
`"Point.h"`

---

**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};  
}

a01f81, 8 lines

"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);
}
```

```
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;
}
```

```
"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();
}
```

"Point.h"	b5562d, 5 lines
-----------	-----------------

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
    P v = b - a;
    return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

"Point.h"	496e9b, 6 lines
-----------	-----------------

```
template<class P>
bool angleCmp(P a, P b) {
    auto half = [](P p) { return sgn(p.y) ? -sgn(p.x); };
    int A = half(a), B = half(b);
    return A == B ? a.cross(b) > 0 : A < B;
}
```

```

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
006002 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 * (11)b.x) <
       make_tuple(b.t, b.half(), a.x * (11)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.t) 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)};
}
```

"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;
}
```

"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;
}
```

**Description:** Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be

```
Point<double>.
"Point.h" e0cfba, 9 lines

template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
    P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
    double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
    if (h2 < 0) return {};
    if (h2 == 0) return {p};
    P h = ab.unit() * sqrt(h2);
    return {p - h, p + h};
}
```

CirclePoly.h

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

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

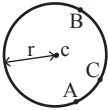
```
".../content/geometry/Point.h" ale63, 19 lines

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



```
"Point.h" 1caa3a, 9 lines

typedef Point<double> P;
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;
}
```

MinEnclosingCircle.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;
}
```

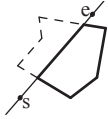
PolygonCut.h

**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;
}
```

PolygonUnion.h

**Description:** Calculates the area of the union of  $n$  polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

**Time:**  $\mathcal{O}(N^2)$ , where  $N$  is the total number of points

```
"Point.h", "sideOf.h" 3931c6, 33 lines


typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
    double ret = 0;
    rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
        P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
        vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
        rep(j,0,sz(poly)) if (i != j) {
            rep(u,0,sz(poly[j])) {
                P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
                int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
                if (sc != sd) {
                    double sa = C.cross(D, A), sb = C.cross(D, B);
                    if (min(sc, sd) < 0)
                        segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
                } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0) {
                    segs.emplace_back(rat(C - A, B - A), 1);
                    segs.emplace_back(rat(D - A, B - A), -1);
                }
            }
        }
        sort(all(segs));
        for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
        double sum = 0;
        int cnt = segs[0].second;
        rep(j,1,sz(segs)) {
            if (!cnt) sum += segs[j].first - segs[j - 1].first;
            cnt += segs[j].second;
        }
        ret += A.cross(B) * sum;
    }
    return ret / 2;
}
```

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

### PointInsideHull.h

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

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

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

### LineHullIntersection.h

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

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

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

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

### HullTangents.h

**Description:** Finds the left and right tangent vertices of a convex polygon relative to point a. The polygon must have at least 3 vertices, be CCW-ordered, and no collinear points. Returns the vertex indices.

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

"Point.h"	249823, 27 lines
<pre> <b>template</b> &lt;class P&gt; pii hullTangents(const vector&lt;P&gt;&amp; p, P a) {     <b>int</b> n = sz(p), t[2];     rep(it, 0, 2) {         <b>auto</b> dir = [&amp;](<b>int</b> i) {             P u = p[i] - a, v = p[(i + 1) % n] - a;             <b>auto</b> c = u.cross(v);             <b>if</b> (c != 0) <b>return</b> c &lt; 0;             <b>if</b> (u.dot(v) &lt;= 0) <b>return</b> true;             <b>return</b> u.dist2() &gt; v.dist2();         };         <b>auto</b> idir = [&amp;](<b>int</b> i) { <b>return</b> dir(i) ^ it; };         <b>if</b> (idir(0) &amp;&amp; !idir(n - 1)) { t[it] = 0; <b>continue</b>; }         <b>int</b> s[2] = {0, n - 1};         <b>while</b> (s[1] - s[0] &gt; 2) {             <b>int</b> mid = (s[0] + s[1]) / 2, x = idir(mid);             <b>if</b> (idir(s[x ^ 1]) == (x ^ 1)) {                 s[x] = mid;             } <b>else</b> {                 <b>bool</b> b = a.cross(p[mid], p[s[1]]) &lt; 0;                 s[b ^ x ^ it ^ 1] = mid;             }         }         t[it] = s[0] + 1 + (idir(s[0] + 1) == 0);     }     <b>return</b> {t[0], t[1]}; } </pre>	

## 8.4 Misc. Point Set Problems

### ClosestPair.h

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

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

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

### ManhattanMST.h

**Description:** Given N points, returns up to 4\*N edges, which are guaranteed to contain a minimum spanning tree for the graph with edge weights  $w(p, q) = -p.x - q.x - + -p.y - q.y -$ . Edges are in the form (distance, src, dst). Use a standard MST algorithm on the result to find the final MST.

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

"Point.h"	df6f59, 23 lines
<pre> <b>typedef</b> Point&lt;<b>int</b>&gt; P; vector&lt;array&lt;<b>int</b>, 3&gt;&gt; manhattanMST(vector&lt;P&gt; ps) {     vi id(sz(ps));     iota(all(id), 0);     vector&lt;array&lt;<b>int</b>, 3&gt;&gt; edges;     rep(k,0,4) {         sort(all(id), [&amp;](<b>int</b> i, <b>int</b> j) {             <b>return</b> (ps[i]-ps[j]).x &lt; (ps[j]-ps[i]).y;});         map&lt;<b>int</b>, <b>int</b>&gt; sweep;         <b>for</b> (<b>int</b> i : id) {             <b>for</b> (<b>auto</b> it = sweep.lower_bound(-ps[i].y);                 it != sweep.end(); sweep.erase(it++)) {                 <b>int</b> j = it-&gt;second;                 P d = ps[i] - ps[j];                 <b>if</b> (d.y &gt; d.x) <b>break</b>;                 edges.push_back({d.y + d.x, i, j});             }             sweep[-ps[i].y] = i;         }         <b>for</b> (P&amp; p : ps) <b>if</b> (k &amp; 1) p.x = -p.x; <b>else</b> swap(p.x, p.y);     }     <b>return</b> edges; } </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"	cefdff5, 88 lines
<pre> <b>typedef</b> Point&lt;ll&gt; P; <b>typedef</b> struct Quad* Q; <b>typedef</b> __int128_t lll; // (can be ll if coords are &lt; 2e4) P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point </pre>	

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

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
    lll 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 > 0;
}

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

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

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

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

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

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

```
}

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

8.5 3D

PolyhedronVolume.h

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

---

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

---

3058c3, 6 lines

Point3D.h

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

---

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

---

8058ae, 32 lines

3dHull.h

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

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

---

"Point3D.h" 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);
```

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

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

```
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;
}
```

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

```

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++);
    }
};

```

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

```

struct SuffixTree {
    enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur position
    int t[N][ALPHA], l[N], r[N], p[N], s[N], v=0, q=0, m=2;

    void ukkadd(int i, int c) { suff:
        if (r[v]<=q) {
            if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
                p[m++]=v; v=s[v]; q=r[v]; goto suff; }
            v=t[v][c]; q=l[v];
        }
    }
}

```

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

```

// example: find longest common substring (uses ALPHA = 28)
pii best;

int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
    rep(c, 0, ALPHA) if (t[node][c] != -1)
        mask |= lcs(t[node][c], i1, i2, len);
    if (mask == 3)
        best = max(best, {len, r[node] - len});
    return mask;
}

static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
}
};

```

```

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

```

```
        return ha[b] - ha[a] * pw[b - a];
    }
};

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

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

## AhoCorasick.h

**Description:** Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(−, word) finds all words (up to  $N\sqrt{N}$  many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries. **Time:** construction takes  $\mathcal{O}(26N)$ , where  $N$  = sum of length of patterns. find(x) is  $\mathcal{O}(N)$ , where  $N$  = length of x. findAll is  $\mathcal{O}(NM)$ .

f35677, 66 lines

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

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

```
    }
    }
}

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

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

## Duval.h

**Description:** Constructs Lyndon factorization of s. A word is called simple iff it is strictly smaller than any of its nontrivial suffixes. The Lyndon factorization of a string is the unique division into non-increasing simple words. **Time:**  $\mathcal{O}(n)$

cab391, 12 lines

```
vi duval(const string& s) {
    int n = sz(s); vi f;
    for (int i = 0; i < n; i) {
        int j = i + 1, k = i;
        for (; j < n && s[k] <= s[j]; j++) {
            if (s[k] < s[j]) k = i;
            else ++k;
        }
        for (; i <= k; i += j - k) f.push_back(i);
    }
    return f.push_back(n), f;
}
```

## Various (10)

### 10.1 Misc. algorithms

#### TernarySearch.h

**Description:** Find the smallest i in  $[a, b]$  that maximizes  $f(i)$ , assuming that  $f(a) < \dots < f(i) \geq \dots \geq f(b)$ . To reverse which of the sides allows non-strict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B). **Usage:** int ind = ternSearch(0,n-1,[&](int i){return a[i];}); **Time:**  $\mathcal{O}(\log(b-a))$

9155b4, 11 lines

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

```
    }
    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

#### KnuthDP.h

**Description:** When doing DP on intervals:  $a[i][j] = \min_{i < k < j} (a[i][k] + a[k][j]) + f(i, j)$ , where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search  $k = p[i][j]$  for  $a[i][j]$  only between  $p[i][j-1]$  and  $p[i+1][j]$ . This is known as Knuth DP. Sufficient criteria for this are if  $f(b, c) \leq f(a, d)$  and  $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$  for all  $a \leq b \leq c \leq d$ . Consider also: LineContainer (ch. Data structures), monotone queues, ternary search. **Time:**  $\mathcal{O}(N^2)$

#### 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 Optimization tricks

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;
}
```