

The University of Austin at Texas

those who know

Aaryan Prakash, Dylan Smith, Mark Wen

SCUSA 2024

November 16, 2024

Contest (1)

```
template.cpp14 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()#define pb push_backtypedef long long ll;typedef pair<int, int> pii;typedef vector<int> vi;int main() {cin.tie(0)->sync_with_stdio(0);}
```

```
.bashrc10 linesrun () {ok=1if [[ ! -f $1 || $1 -ot $1.cpp ]] theng++ $1.cpp -O2 -o $1 -std=c++17 -Wall -Wextra -Wshadow -Wconversion -fsanitize=undefined,address || ok=0fi[[ $ok -eq 1 ]] && ./$1}xmodmap -e 'clear Lock' -e 'keycode 0x42 = Escape'
```

```
.vimrc5 linesset cin aw ai is ts=4 sw=4 tm=50 rnu noeb bg=dark ru cul mouse=asy on | no ;:" Select region and then type :Hash to hash your selection.ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \ \| md5sum \| cut -c-6
```

```
brute.sh12 lines#!/bin/zshsz=100for ((i=1;;i++)); doecho "$i"./gen "$i" "$sz" > input./sol < input > output1./brute < input > output2if (! diff output1 output2); thenbreakfiendone
```

Mathematics (2)

2.1 Equations

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

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

2.2 Geometry

2.2.1 Triangles

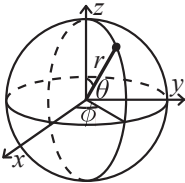
Circumradius: $R = abc/4A$
Inradius: $r = A/p$
Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$
Length of bisector (divides angles in two):

$$s_a = \sqrt{bc \left[1 - \left(\frac{a}{b+c} \right)^2 \right]}$$

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 Spherical coordinates



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

2.3 Probability theory

$$\begin{aligned} \sigma^2 &= V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 \\ \mathbb{E}(aX + bY) &= a\mathbb{E}(X) + b\mathbb{E}(Y) \\ \text{ind. } X, Y, V(aX + bY) &= a^2V(X) + b^2V(Y). \end{aligned}$$

2.3.1 Discrete distributions

Binomial distribution

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$
$$\mu = np, \sigma^2 = np(1-p)$$

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

Geometric distribution

$$p(k) = p(1-p)^{k-1}, \quad k = 1, 2, \dots$$
$$\mu = \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}$$

Poisson distribution

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

2.3.2 Continuous distributions

Uniform distribution

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$
$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$
$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

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

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

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

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

2.4 Markov chains

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

π is a stationary distribution if $\pi = \pi \mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i . π_j/π_i is the expected number of visits in state j between two visits in state i .

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i 's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$.

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

Data structures (3)

OrderStatisticTree.h

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

<pre>#include <bits/extc++.h> //893 using namespace __gnu_pbds; template<class T> using Tree = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>; //988 void example() { Tree<int> t, t2; t.insert(8); auto it = t.insert(10).first; assert(it == t.lower_bound(9)); //6bd assert(t.order_of_key(10) == 1); assert(t.order_of_key(11) == 2); assert(*t.find_by_order(0) == 8); t.join(t2); // assuming T< T2 or T> T2, merge t2 into t } //cbb</pre>	782797, 16 lines
--	------------------

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).

<pre>#include <bits/extc++.h> //1e4 // To use most bits rather than just the lowest ones: struct chash { // large odd number for C const uint64_t C = 11(4e18 * acos(0)) 71; ll operator()(ll x) const { return __builtin_bswap64(x*C) ; } } //198 __gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{ 1<<16});</pre>	d77092, 7 lines
---	-----------------

Matrix.h

Description: Basic operations on square matrices.
Usage: Matrix<int, 3> A;
A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};
vector<int> vec = {1,2,3};
vec = (A^N) * vec;

<pre>template<class T, int N> struct Matrix { //1aa typedef Matrix M; array<array<T, N>, N> d{}; M operator*(const M& m) const { M a; rep(i,0,N) rep(j,0,N) //683 rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];</pre>	c43c7d, 26 lines
--	------------------

<pre> return a; } vector<T> operator*(const vector<T>& vec) const { vector<T> ret(N); //9bd rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j]; return ret; } M operator^(ll p) const { assert(p >= 0); //358 M a, b(*this); rep(i,0,N) a.d[i][i] = 1; while (p) { if (p&1) a = a*b; b = b*b; //1d8 p >>= 1; } return a; } } //214</pre>	
--	--

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

<pre>struct Line { //7e3 mutable ll k, m, p; bool operator<(const Line& o) const { return k < o.k; } bool operator<(ll x) const { return p < x; } }; //d77 struct LineContainer : multiset<Line, less<>> { // (for doubles, use inf = 1/.0, div(a,b) = a/b) static const ll inf = LLONG_MAX; ll div(ll a, ll b) { // floored division return a / b - ((a ^ b) < 0 && a % b); } //66e bool isect(iterator x, iterator y) { if (y == end()) return x->p = inf, 0; if (x->k == y->k) x->p = x->m > y->m ? inf : -inf; else x->p = div(y->m - x->m, x->k - y->k); return x->p >= y->p; //bec } void add(ll k, ll m) { auto z = insert({k, m, 0}), y = z++, x = y; while (isect(y, z)) z = erase(z); if (x != begin() && isect(--x, y)) isect(x, y = erase(y)); //890 while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y)); } ll query(ll x) { assert(!empty()); //b07 auto l = *lower_bound(x); return l.k * x + l.m; } } };</pre>	8ec1e7, 30 lines
---	------------------

Treap.h

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

<pre>struct Node { //e9f Node *l = 0, *r = 0; int val, y, c = 1; Node(int v) : val(v), y(rand()) {} void recalc(); } //3ef</pre>	30f532, 55 lines
--	------------------

<pre>int cnt(Node* n) { return n ? n->c : 0; } void Node::recalc() { c = cnt(l) + cnt(r) + 1; } template<class F> void each(Node* n, F f) { //5d5 if (n) { each(n->l, f); f(n->val); each(n->r, f); } } pair<Node*, Node*> split(Node* n, int k) { if (!n) return {}; //ca5 if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k) auto pa = split(n->l, k); n->l = pa.second; n->recalc(); return {pa.first, n}; //b54 } else { auto pa = split(n->r, k - cnt(n->l) - 1); // and just " k" n->r = pa.first; n->recalc(); return {n, pa.second}; //86d } }</pre>	
--	--

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

<pre>Node* ins(Node* t, Node* n, int pos) { auto pa = split(t, pos); return merge(merge(pa.first, n), pa.second); //99b } // Example application: move the range [l, r) to index k void move(Node& t, int l, int r, int k) { Node *a, *b, *c; //99c tie(a,b) = split(t, l); tie(b,c) = split(b, r - l); if (k <= l) t = merge(ins(a, b, k), c); else t = merge(a, ins(c, b, k - r)); }</pre>	
--	--

FenwickTree.h

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

<pre>struct FT { //711 vector<ll> s; FT(int n) : s(n) {} void update(int pos, ll dif) { // a[pos] += dif for (; pos < sz(s); pos = pos + 1) s[pos] += dif; } //cc4 ll query(int pos) { // sum of values in [0, pos) ll res = 0; for (; pos > 0; pos &= pos - 1) res += s[pos-1]; return res; } //477 int lower_bound(ll sum) { // min pos st sum of [0, pos] >= sum // Returns n if no sum is >= sum, or -1 if empty sum is .</pre>	e62fac, 22 lines
--	------------------

```
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) { //fc5
        if (pos + pw <= sz(s) && s[pos + pw - 1] < sum)
            pos += pw, sum -= s[pos - 1];
    }
    return pos;
} //e03
};
```

FenwickTree2d.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).

Time: $\mathcal{O}(\log^2 N)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

```
"FenwickTree.h" e2f703, 22 lines
struct FT2 { //4ce
    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].pb(y);
    } //57f
    void init() {
        for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
    }
    int ind(int x, int y) {
        return (int)(lower_bound(all(ys[x]), y) - ys[x].begin())
        ); } //358
    void update(int x, int y, ll dif) {
        for (; x < sz(ys); x |= x + 1)
            ft[x].update(ind(x, y), dif);
    }
    ll query(int x, int y) { //688
        ll sum = 0;
        for (; x; x &= x - 1)
            sum += ft[x - 1].query(ind(x - 1, y));
        return sum;
    } //e03
};
```

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.

Usage: RMQ rmq(values);
rmq.query(inclusive, exclusive);

Time: $\mathcal{O}(|V| \log |V| + Q)$

```
template<class T> //722
struct RMQ {
    vector<vector<T>>> jmp;
    RMQ(const vector<T>& V) : jmp(1, V) {
        for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k)
            jmp.emplace_back(sz(V) - pw * 2 + 1); //f6c
        rep(j, 0, sz(jmp[k]))
            jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
    }
    T query(int a, int b) { //a3d
        assert(a < b); // or return inf if a == b
        int dep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
    }
} //214
```

MoQueries.h

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

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

```
a12ef4, 49 lines
void add(int ind, int end) { ... } // add a[ind] (end = 0
or 1) //342
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer

vi mo(vector<pii> Q) {
    int L = 0, R = 0, blk = 350; // ~N/sqrt(Q) //cb0
    vi s(sz(Q)), res = s;
    #define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1)
    )
    iota(all(s), 0);
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]);
    });
    for (int qi : s) { //623
        pii q = Q[qi];
        while (L > q.first) add(--L, 0);
        while (R < q.second) add(R++, 1);
        while (L < q.first) del(L++, 0);
        while (R > q.second) del(--R, 1); //d22
        res[qi] = calc();
    }
    return res;
} //842
vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root
= 0) {
    int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
    vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
    add(0, 0), in[0] = 1;
    auto dfs = [&](int x, int p, int dep, auto& f) -> void {
        //263
        par[x] = p;
        L[x] = N;
        if (dep) I[x] = N++;
        for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
        if (!dep) I[x] = N++; //23e
        R[x] = N;
    };
    dfs(root, -1, 0, dfs);
    #define K(x) pii(I[x][0] / blk, I[x][1] ^ -(I[x][0] / blk &
    1))
    iota(all(s), 0); //064
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]);
    });
    for (int qi : s) rep(end, 0, 2) {
        int &a = pos[end], b = Q[qi][end], i = 0;
        #define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
        else { add(c, end); in[c] = 1; } a = c; } //440
        while (!(L[b] <= L[a] && R[a] <= R[b]))
            I[i++] = b, b = par[b];
        while (a != b) step(par[a]);
        while (i--) step(I[i]);
        if (end) res[qi] = calc(); //695
    }
    return res;
}
```

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

```
84e98b, 21 lines
struct RollbackUF { //f73
```

```
vi e; vector<pii> st;
RollbackUF(int n) : e(n, -1) {}
int size(int x) { return -e[find(x)]; }
int find(int x) { return e[x] < 0 ? x : find(e[x]); }
int time() { return sz(st); } //cbd
void rollback(int t) {
    for (int i = time(); i --> t;)
        e[st[i].first] = st[i].second;
    st.resize(t);
} //e73
bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.pb({a, e[a]}); //0d8
    st.pb({b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
}
}; //214
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

c9b7b0, 17 lines

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

PolyRoots.h

Description: Finds the real roots to a polynomial.

Usage: polyRoots({{2,-3,1}}, -1e9, 1e9) // solve x²-3x+2 = 0

Time: $\mathcal{O}(n^2 \log(1/\epsilon))$

"Polynomial.h" fbf066, 23 lines

```
vector<double> polyRoots(Poly p, double xmin, double xmax)
{ //840
    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
    vector<double> ret;
    Poly der = p;
    der.diff();
    auto dr = polyRoots(der, xmin, xmax); //ec1
    dr.pb(xmin - 1);
    dr.pb(xmax + 1);
    sort(all(dr));
    rep(i, 0, sz(dr) - 1) {
        double l = dr[i], h = dr[i + 1]; //189
        bool sign = p(l) > 0;
        if (sign ^ (p(h) > 0)) {
            rep(it, 0, 60) { // while (h - l > 1e-8)
                double m = (l + h) / 2, f = p(m);
                if ((f <= 0) ^ sign) l = m; //810
                else h = m;
            }
        }
    }
}
```

```
        ret.pb((1 + h) / 2);
    }
} //808
return ret;
}
```

PolyInterpolate.h

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

```
08bf48, 13 lines
typedef vector<double> vd; //159
vd interpolate(vd x, vd y, int n) {
    vd res(n), temp(n);
    rep(k,0,n-1) rep(i,k+1,n)
        y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0; temp[0] = 1; //746
    rep(k,0,n) rep(i,0,n) {
        res[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
    } //0e1
    return res;
}
```

BerlekampMassey.h

Description: Recovers any n -order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$. **Usage:** berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2} **Time:** $\mathcal{O}(N^2)$

```
96548b, 20 lines
.../number-theory/ModPow.h"
vector<ll> berlekampMassey(vector<ll> s) { //b21
    int n = sz(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;

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

LinearRecurrence.h

Description: Generates the k 'th term of an n -order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \dots \geq n-1]$ 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:** $\mathcal{O}(n^2 \log k)$

```
f4e444, 26 lines
typedef vector<ll> Poly; //bb1
ll linearRec(Poly S, Poly tr, ll k) {
    int n = sz(tr);

    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1); //251
        rep(i,0,n+1) rep(j,0,n+1)
            res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
    }
```

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

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

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

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

4.2 Optimization

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is ϵ 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:** $\mathcal{O}(\log((b-a)/\epsilon))$

```
d7b114, 15 lines
template<class F> //5c6
double gss(double a, double b, F f) {
    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) //905
        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; //00c
            x2 = a + r*(b-a); f2 = f(x2);
        }
    return a;
}
```

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

IntegrateAdaptive.h

Description: Fast integration using an adaptive Simpson's rule. **Usage:** double sphereVolume = quad(-1, 1, [](double x) { return quad(-1, 1, [&](double y) { return quad(-1, 1, [&](double z) { return x*x + y*y + z*z < 1; }}}); }); **typedef double d;** //e70 **#define** S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6

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

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \leq b, x \geq 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable. **Usage:** vvd A = {{1,-1}, {-1,1}, {-1,-2}}; vd b = {1,1,-4}, c = {-1,-1}, x; T val = LPSolver(A, b, c).solve(x); **Time:** $\mathcal{O}(NM * \#pivots)$, 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 + modkP
>... //629
typedef vector<T> vd;
typedef vector<vd> vvd;

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

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

    LPSolver(const vvd& A, const vd& b, const vd& c) :
        m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
            rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j]; //108
            rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
            rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
            N[n] = -1; D[m+1][n] = 1;
        }
    //9c3
    void pivot(int r, int s) {
        T *a = D[r].data(), inv = 1 / a[s];
        rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
            T *b = D[i].data(), inv2 = b[s] * inv;
            rep(j,0,n+2) b[j] -= a[j] * inv2; //d0d
            b[s] = a[s] * inv2;
        }
        rep(j,0,n+2) if (j != s) D[r][j] *= inv;
        rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv; //aa5
        swap(B[r], N[s]);
    }

    bool simplex(int phase) {
        int x = m + phase - 1; //c51
        for (;) {
            int s = -1;
            rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
        }
```

```

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

4.3 Matrices

Determinant.h

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

```

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

IntDeterminant.h

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

```

const ll mod = 12345; //cab
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step//c65
                ll t = a[i][i] / a[j][i];
                if (t) rep(k,i,n)
                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1; //bc6
            }
        }
    }
    ans = ans * a[i][i] % mod;
```

```

    if (!ans) return 0;
} //b19
return (ans + mod) % mod;
}
```

SolveLinear.h

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

```

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

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

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

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

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

```

"SolveLinear.h"
08e495, 7 lines

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

SolveLinearBinary.h

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

```

typedef bitset<1000> bs; //d90

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
```

```

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

MatrixInverse.h

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

```

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

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

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

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

MatrixInverse-mod.h

Description: Invert matrix A modulo a prime. Returns rank; result is stored in A unless singular (rank < n). For prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of $A \bmod p$, and k is doubled in each step.

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

```
"/number-theory/ModPow.h"                                0b7b13, 37 lines

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

    rep(i,0,n) { //79d
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n) if (A[j][k]) {
            r = j; c = k; goto found;
        }
        return i; //4e3
    }
found:
    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]); //416
    ll v = modpow(A[i][i], mod - 2);
    rep(j,i+1,n) {
        ll f = A[j][i] * v % mod;
        A[j][i] = 0;
        rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod; //9f7
        rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % mod;
    }
    rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
    rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1; //e3d

    for (int i = n-1; i > 0; --i) rep(j,0,i) {
        ll v = A[j][i];
        rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) % mod; //4b2
    }

    rep(i,0,n) rep(j,0,n)
        A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0) * mod;
    return n; //400
}
```

Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

Fails if the solution is not unique.

If $|d_i| > |p_i| + |q_{i-1}|$ for all i , or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.

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

```
8f9fa8, 26 lines

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

4.4 Fourier transforms

FastFourierTransform.h

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

Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$)

```
00ced6, 35 lines

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

```
vd res(sz(a) + sz(b) - 1);
int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
vector<C> in(n), out(n); //d93
copy(all(a), begin(in));
rep(i,0,sz(b)) in[i].imag(b[i]);
fft(in);
for (C& x : in) x *= x;
rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]); //36e
fft(out);
rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
return res;
}
```

FastFourierTransformMod.h

Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.

Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)

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

typedef vector<ll> vl; //2c4
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vl res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
    ;
    vector<C> L(n), R(n), outs(n), outl(n); //c4f
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    ;
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    ;
    fft(L), fft(R);
    rep(i,0,n) {
        int j = -i & (n - 1); //3eb
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / li;
    }
    fft(outl), fft(outs);
    rep(i,0,sz(res)) { //58f
        ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
        ;
        ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res; //510
}
```

NumberTheoreticTransform.h

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

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

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

const ll mod = (119 << 23) + 1, root = 62; // =
    998244353; //0ca
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 <<
    21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n); //cc5
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod; //4a0
```

```
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)//ed7
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j
                ];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        }//dfc
    }
    vl conv(const vl &a, const vl &b) {
        if (a.empty() || b.empty()) return {};
        int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
            n = 1 << B;//d58
        int inv = modpow(n, mod - 2);
        vl L(a), R(b), out(n);
        L.resize(n), R.resize(n);
        ntt(L), ntt(R);
        rep(i,0,n)//f18
            out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
        ntt(out);
        return {out.begin(), out.begin() + s};
    }
```

FastSubsetTransform.h

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

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

464cf3, 16 lines

```
void FST(vi& a, bool inv) {//ae8
    for (int n = sz(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
            int &u = a[j], &v = a[j + step]; tie(u, v) =
                inv ? pii(v - u, u) : pii(v, u + v); // AND
                inv ? pii(v, u - v) : pii(u + v, u); // OR//0af
                pii(u + v, u - v); // XOR
        }
        if (inv) for (int& x : a) x /= sz(a); // XOR only
    }//dc4
    vi conv(vi a, vi b) {
        FST(a, 0); FST(b, 0);
        rep(i,0,sz(a)) a[i] *= b[i];
        FST(a, 1); return a;
    }//cbb
```

4.5 Polynomial

PolyBase.h

Description: A FFT based Polynomial class.

"...number-theory/ModularArithmetic.h", "FastFourierTransform.h",
"FastFourierTransformMod.h", "NumberTheoreticTransform.h" dd1be7, 35 lines

```
typedef Mod num;//810
typedef vector<num> poly;
poly &operator+=(poly &a, const poly &b) {
    a.resize(max(sz(a), sz(b)));
    rep(i, 0, sz(b)) a[i] = a[i] + b[i];
    return a;//8a9
}
poly &operator==(poly &a, const poly &b) {
    a.resize(max(sz(a), sz(b)));
    rep(i, 0, sz(b)) a[i] = a[i] - b[i];
    return a;//e10
}
```

```
poly &operator*=(poly &a, const poly &b) {
    if (sz(a) + sz(b) < 100){
        poly res(sz(a) + sz(b) - 1);//025
```

```
        rep(i,0,sz(a)) rep(j,0,sz(b))
            res[i + j] = (res[i + j] + a[i] * b[j]);
        return (a = res);
    }
    // auto res = convMod<mod>(vl(all(a)), vl(all(b)));//0cb
    auto res = conv(vl(all(a)), vl(all(b)));
    return (a = poly(all(res)));
}
poly operator*(poly a, const num b) {
    poly c = a;//41c
    for(auto& i : c) i = i * b;
    return c;
}
#define OP(o, oe) \
    poly operator o(poly a, poly b) { \\\f19
        poly c = a; \
        return c o##= b; \
    }
OP(*, *=) OP(+, +=) OP(-, -=);
```

PolyEvaluate.h

Description: Multi-point evaluation. Evaluates a given polynomial A at $A(x_0), \dots A(x_n)$.

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

"PolyBase.h", "PolyMod.h" dc2cdf, 14 lines

```
vector<num> eval(const poly &a, const vector<num> &x) {\f9
    fa
    int n = sz(x);
    if (!n) return {};
    vector<poly> up(2 * n);
    rep(i, 0, n) up[i + n] = poly({num(0) - x[i], 1});
    for (int i = n - 1; i > 0; i--)//923
        up[i] = up[2 * i] * up[2 * i + 1];
    vector<poly> down(2 * n);
    down[1] = a % up[1];
    rep(i, 2, 2 * n) down[i] = down[i / 2] % up[i];
    vector<num> y(n);//835
    rep(i, 0, n) y[i] = down[i + n][0];
    return y;
}
```

PolyIntegDeriv.h

Description: Calculate $\frac{da}{dx}$ and $\int a dx$.

Time: $\mathcal{O}(|a|)$

"PolyBase.h" 803fd5, 14 lines

```
poly deriv(poly a) {\faba
    if (a.empty()) return {};
    poly b(sz(a) - 1);
    rep(i, 1, sz(a)) b[i - 1] = a[i] * num(i);
    return b;
} //6f9
poly integr(poly a) {
    if (a.empty()) return {0};
    poly b(sz(a) + 1);
    b[1] = num(1);
    rep(i, 2, sz(b)) b[i] = b[mod%i]*Mod(-mod/i+mod);//176
    rep(i, 1, sz(b)) b[i] = a[i-1] * b[i];
    return b;
}
```

PolyInterpolate2.h

Description: Given n points $(x[i], y[i])$, computes an $n-1$ -degree polynomial p that passes through them: $p(x) = a[0] \cdot x^0 + \dots + a[n-1] \cdot x^{n-1}$.

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

"PolyBase.h", "PolyIntegDeriv.h", "PolyEvaluate.h" b911f5, 11 lines

```
poly interp(vector<num> x, vector<num> y) {\f74d
    int n=sz(x);
    vector<poly> up(n*2);
    rep(i,0,n) up[i+n] = poly({num(0)-x[i], num(1)});
```

```
    for(int i=n-1; i>0;i--) up[i] = up[2*i]*up[2*i+1];
    vector<num> a = eval(deriv(up[1]), x);//6dd
    vector<poly> down(2*n);
    rep(i,0,n) down[i+n] = poly({y[i]*(num(1)/a[i])});
    for(int i=n-1;i>0;i--) down[i] = down[i*2] * up[i*2+1] +
        down[i*2+1] * up[i*2];
    return down[1];
} //cbb
```

PolyInverse.h

Description: Calculate the first $|a|$ coefficients of a^{-1} .

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

"PolyBase.h" 703c16, 7 lines

```
poly modK(poly a, int k) { return {a.begin(), a.begin() +
    min(k, sz(a))}; } //140
poly inverse(poly A) {
    poly B = poly({num(1) / A[0]});
    while (sz(B) < sz(A))
        B = modK(B * (poly({num(2)}) - modK(A, 2*sz(B)) * B), 2
            * sz(B));
    return modK(B, sz(A)); //556
}
```

PolyLogExp.h

Description: Calculate the first $|a|$ coefficients of a^{-1} of $\log a$ and $\exp a$.

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

"PolyBase.h", "PolyInverse.h", "PolyIntegDeriv.h" 83ea75, 14 lines

```
poly log(poly a) {\f9c1
    return modK(integr(deriv(a) * inverse(a)), sz(a));
}
poly exp(poly a) {
    poly b(1, num(1));
    if (a.empty())//8ff
        return b;
    while (sz(b) < sz(a)) {
        b.resize(sz(b) * 2);
        b *= (poly({num(1)}) + modK(a, sz(b)) - log(b));
        b.resize(sz(b) / 2 + 1); //1f2
    }
    return modK(b, sz(a));
}
```

PolyMod.h

Description: Calculate the remainder and quotient of the Euclidean division $\frac{a}{b}$.

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

"PolyBase.h", "PolyInverse.h" 264551, 20 lines

```
poly &operator/=(poly &a, poly b) {\fb9b
    if (sz(a) < sz(b))
        return a = {};
    int s = sz(a) - sz(b) + 1;
    reverse(all(a)), reverse(all(b));
    a.resize(s), b.resize(s); //e15
    a = a * inverse(b);
    a.resize(s), reverse(all(a));
    return a;
}
OP(/, /=) //9fe
poly &operator%=(poly &a, poly b) {
    if (sz(a) < sz(b))
        return a;
    poly c = (a / b) * b;
    a.resize(sz(b) - 1); //f62
    rep(i, 0, sz(a)) a[i] = a[i] - c[i];
    return a;
}
OP(%, %=)
```


PolyPow.h

Description: Calculate the first $|a|$ coefficients of a^m .
Time: $\mathcal{O}(n \log n)$.

"PolyBase.h", "PolyLogExp.h"	f0005c, 13 lines
poly pow(poly a, ll m) { <i>//760</i> int p = 0, n = sz(a); while (p < sz(a) && a[p].v == 0) ++p; if (ll(m)*p >= sz(a)) return poly(sz(a)); num j = a[p]; <i>//a78</i> a = {a.begin() + p, a.end()}; a = a * (num(1) / j); a.resize(n); auto res = exp(log(a) * num(m)) * (j ^ m); res.insert(res.begin(), p*m, 0); <i>//6a6</i> return {res.begin(), res.begin()+n}; }	

PolyRoots.h

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

"Polynomial.h"	fbf066, 23 lines
vector<double> polyRoots(Poly p, double xmin, double xmax) { <i>//840</i> 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); <i>//ec1</i> dr.pb(xmin-1); dr.pb(xmax+1); sort(all(dr)); rep(i,0,sz(dr)-1) { double l = dr[i], h = dr[i+1]; <i>//189</i> bool sign = p(l) > 0; if (sign ^ (p(h) > 0)) { rep(it,0,60) { <i>// while (h - l > 1e-8)</i> double m = (l + h) / 2, f = p(m); if ((f <= 0) ^ sign) l = m; <i>//810</i> else h = m; } ret.pb((l + h) / 2); } } <i>//808</i> return ret; }	

Number theory (5)

5.1 Modular arithmetic

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM ≤ mod and that mod is a prime.

ll* inv = new ll[LIM] - 1; inv[1] = 1; <i>//b4a</i> rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;	b4a981, 2 lines
--	-----------------

ModPow.h

const int mod = 1000000007; <i>// faster if const//dce</i>	7b7908, 8 lines
ll modpow(ll b, ll e) { ll ans = 1; for (; e; b = b * b % mod, e /= 2) if (e & 1) ans = ans * b % mod; <i>//7e5</i> 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) { <i>//260</i> 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; <i>//d16</i> 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; } <i>//cbb</i>	

ModSum.h

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

typedef unsigned long long ull; <i>//df3</i> ull sumsq(ull to) { return to / 2 * ((to-1) 1); }	5c5bc5, 16 lines
ull divsum(ull to, ull c, ull k, ull m) { ull res = k / m * sumsq(to) + c / m * to; k %= m; c %= m; <i>//e1a</i> if (!k) return res; ull to2 = (to * k + c) / m; return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k); } <i>//1ae</i> ll modsum(ull to, ll c, ll k, ll m) { c = ((c % m) + m) % m; k = ((k % m) + m) % m; return to * c + k * sumsq(to) - m * divsum(to, c, k, m); } <i>//cbb</i>	

ModMulLL.h

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

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

ModSqrt.h

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

"ModPow.h"	19a793, 24 lines
ll sqrt(ll a, ll p) { <i>//473</i> a %= p; if (a < 0) a += p; if (a == 0) return 0; assert(modpow(a, (p-1)/2, p) == 1); <i>// else no solution</i> if (p % 4 == 3) return modpow(a, (p+1)/4, p); <i>// a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5//a48</i> 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; <i>//c4b</i> 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) <i>//faf</i> 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; <i>//a28</i> b = b * g % p; } }	
---	--

5.2 Primality

FastEratosthenes.h

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

const int LIM = 1e6; <i>//058</i> 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; <i>//86b</i> for (int i = 3; i <= S; i += 2) if (!sieve[i]) { cp.pb({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) { <i>//62d</i> 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.pb((L + i) * 2 + 1); <i>//0b9</i> } for (int i : pr) isPrime[i] = 1; return pr; }	9ac0a0, 20 lines
---	------------------

MillerRabin.h

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

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

Factor.h

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

Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.
<div>"ModMulLL.h", "MillerRabin.h"21a173, 18 lines</div>
<pre>ull pollard(ull n) {<i>//c81</i> ull x = 0, y = 0, t = 30, prd = 2, i = 1, q; auto f = [&](ull a) { return modmul(a, a, 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; <i>//049</i> x = f(x), y = f(f(y)); } return __gcd(prd, n); } vector<ull> factor(ull n) {<i>//c19</i> if (n == 1) return {}; if (isPrime(n)) return {n}; ull x = pollard(n); auto l = factor(x), r = factor(n / x); l.insert(l.end(), all(r));<i>//363</i> return l; }</pre>

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 <code>__gcd</code> instead. If a and b are coprime, then x is the inverse of $a \pmod b$.
33ba8f, 5 lines

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

CRT.h
Description: Chinese Remainder Theorem. <code>crt(a, m, b, n)</code> computes x such that $x \equiv a \pmod m$, $x \equiv b \pmod n$. If $ a \leq m$ and $ b < n$, x will obey $0 \leq x < \text{lcm}(m,n)$. Assumes $mn < 2^{62}$.
Time: $\log(n)$
<div>"euclid.h"04d93a, 7 lines</div>
<pre>ll crt(ll a, ll m, ll b, ll n) {<i>//eae</i> if (n > m) swap(a, b), swap(m, n); ll x, y, g = euclid(m, n, x, y); assert((a - b) % g == 0); <i>// else no solution</i> x = (b - a) % n * x % n / g * m + a; return x < 0 ? x + m*n/g : x;<i>//6ac</i> }</pre>

5.3.1 Bézout’s identity

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

$$ax+by=d$$

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

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

phiFunction.h
Description: Euler’s ϕ function is defined as $\phi(n) := \#$ of positive integers $\leq n$ that are coprime with n . $\phi(1) = 1, p$ prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1}p_2^{k_2}...p_r^{k_r}$ then $\phi(n) = (p_1-1)p_1^{k_1-1}...(p_r-1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p n}(1-1/p)$. $\sum_{d n}\phi(d) = n, \sum_{1\leq k\leq n, \gcd(k,n)=1}k = n\phi(n)/2, n > 1$

Euler’s thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.
Fermat’s little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p \ \forall a$.
cf7d6d, 8 lines
<pre>const int LIM = 5000000;<i>//70b</i> int phi[LIM]; void calculatePhi() { rep(i,0,LIM) phi[i] = i&1 ? i : i/2; for (int i = 3; i < LIM; i += 2) if(phi[i] == i)<i>//103</i> for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i; } 5.4 Fractions ContinuedFractions.h Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with $p, q \leq N$. It will obey $p/q - x \leq 1/qN$. For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. ($p_k/q_k$ alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a’s eventually become cyclic. Time: $\mathcal{O}(\log N)$ ea42e2, 21 lines typedef double d; <i>// for N ~ 1e7; long double for N ~ 1e9</i> <i>//32b</i> pair<ll, ll> approximate(d x, ll N) { ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x; ; for (;) { ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf), a = (ll)floor(y), b = min(a, lim),<i>//5ad</i> NP = b*P + LP, NQ = b*Q + LQ; if (a > b) { <i>// If b > a/2, we have a semi-convergent that gives us a</i> <i>// better approximation; if b = a/2, we *may* have one.</i> <i>// Return {P, Q} here for a more canonical approximation.</i><i>//fcb</i> return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q) ? make_pair(NP, NQ) : make_pair(P, Q); } if (abs(y = 1/(y - (d)a)) > (d)N*3) { return {NP, NQ};<i>//5c7</i> } LP = P; P = NP; LQ = Q; Q = NQ; } }<i>//cbb</i></pre>

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: <code>fracBS([](Frac f) { return f.p>=3*f.q; }, 10);</code> <i>//</i> {1,3}
Time: $\mathcal{O}(\log(N))$
27ab3e, 25 lines

<pre>struct Frac { ll p, q; };<i>//386</i> template<class F> Frac fracBS(F f, ll N) { bool dir = 1, A = 1, B = 1; Frac lo{0, 1}, hi{1, 1}; <i>// Set hi to 1/0 to search (0, N</i> <i>//262</i> if (f(lo)) return lo; assert(f(hi)); while (A B) { ll adv = 0, step = 1; <i>// move hi if dir, else lo</i> for (int si = 0; step; (step *= 2) >= si) {<i>//7e2</i></pre>

<pre> 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; }<i>//bf0</i> } hi.p += lo.p * adv; hi.q += lo.q * adv; dir = !dir; swap(lo, hi);<i>//f58</i> A = B; B = !adv; } return dir ? hi : lo; }</pre>
--

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), 3144353997927 (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 Mobius Function

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

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

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

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

6.1.2 Derangements

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

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

6.1.3 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.

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

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

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	$\sim 2\text{e}5$	$\sim 2\text{e}8$

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

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

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

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

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

- 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 Network flow

PushRelabel.h

Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.

Time: $\mathcal{O}(V^2 \sqrt{E})$

	91e61c, 48 lines
<pre>struct PushRelabel {<i>//d82</i> struct Edge { int dest, back; ll f, c; }; vector<vector<Edge>> g;<i>//bef</i> vector<ll> ec; vector<Edge*> cur; vector<vi> hs; vi H; PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {} <i>//a7b</i> void addEdge(int s, int t, ll cap, ll rcap=0) { if (s == t) return; g[s].pb({t, sz(g[t]), 0, cap}); g[t].pb({s, sz(g[s])-1, 0, rcap}); }<i>//7a1</i> void addFlow(Edge& e, ll f) { Edge &back = g[e.dest][e.back]; if (!ec[e.dest] && f) hs[H[e.dest]].pb(e.dest); e.f += f; e.c -= f; ec[e.dest] += f;<i>//124</i> back.f -= f; back.c += f; ec[back.dest] -= f; } ll calc(int s, int t) { int v = sz(g); H[s] = v; ec[t] = 1; vi co(2*v); co[0] = v-1;<i>//a96</i> rep(i,0,v) cur[i] = g[i].data(); for (Edge& e : g[s]) addFlow(e, e.c); for (int hi = 0;;) { while (hs[hi].empty()) if (!hi--) return -ec[s];<i>//e2e</i> int u = hs[hi].back(); hs[hi].pop_back(); while (ec[u] > 0) <i>// discharge u</i> if (cur[u] == g[u].data() + sz(g[u])) { H[u] = 1e9; for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]]+1)<i>//9ff</i> H[u] = H[e.dest]+1, cur[u] = &e; if (++co[H[u]], !--co[hi] && hi < v) rep(i,0,v) if (hi < H[i] && H[i] < v) --co[H[i]], H[i] = v + 1; hi = H[u];<i>//7ed</i> } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1) addFlow(*cur[u], min(ec[u], cur[u]->c)); else ++cur[u]; } }<i>//a5b</i> bool leftOfMinCut(int a) { return H[a] >= sz(g); } };</pre>	

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.

<i><bits/extc++.h></i>	bb147c, 77 lines
<pre>const ll INF = numeric_limits<ll>::max() / 4;<i>//d4e</i> struct MCMF { struct edge { int from, to, rev; ll cap, cost, flow;<i>//309</i> }; int N;</pre>	

```
vector<vector<edge>> ed;
vi seen;
vector<ll> dist, pi;//16a
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) { //a9c
    if (from == to) return;
    ed[from].pb(edge{ from,to,sz(ed[to]),cap,cost,0 });
    ed[to].pb(edge{ to,from,sz(ed[from])-1,0,-cost,0 });
}
//635
void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;

//17c
    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({ 0, s });

    while (!q.empty()) { //95a
        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]) { //c63
                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 //dbc
                    q.modify(its[e.to], { -dist[e.to], e.to });
            }
        }
        rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF); //02d
    }

pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) { //aa6
        ll fl = INF;
        for (edge* x = par[t]; x; x = par[x->from])
            fl = min(fl, x->cap - x->flow);

        totflow += fl; //21b
        for (edge* x = par[t]; x; x = par[x->from]) {
            x->flow += fl;
            ed[x->to][x->rev].flow -= fl;
        }
    } //cd4
    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: //7c7
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) //42d
            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
} //e03
```

```
};

GlobalMinCut.h
Description: Find a global minimum cut in an undirected graph, as
represented by an adjacency matrix.
Time: O(V^3)
8b0e19, 21 lines

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

GomoryHu.h
Description: Given a list of edges representing an undirected flow
graph, returns edges of the Gomory-Hu tree. The max flow between any
pair of vertices is given by minimum edge weight along the Gomory-Hu
tree path.
Time: O(V) Flow Computations
"PushRelabel.h"
1ec6c8, 13 lines

struct Edge { int from, to; ll cap; }; //81a
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
    vector<Edge> tree;
    vi par(N);
    rep(i,1,N) {
        PushRelabel D(N); // Dinic also works //489
        for (Edge t : ed) D.addEdge(t.from, t.to, t.cap, t.cap);
        tree.pb({i, par[i], D.calc(i, par[i])});
        rep(j,i+1,N)
            if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
    } //eec
    return tree;
}
```

```
FlowWithDemands.h
Description: Add a new source s' and sink t', new edges from s' to
everything, and new edges from everything to t'. Define:

    • c'((s', v)) = ∑_{u ∈ V} d((u, v)) for each edge (s', v)

    • c'((v, t')) = ∑_{w ∈ V} d((v, w)) for each edge (v, t')

    • c'((u, v)) = c((u, v)) - d((u, v)) for each old edge (u, v)

    • c'((t, s)) = ∞

d41d8c, 1 lines
//d41

7.2 Matching
hopcroftKarp.h
```

```
Description: Fast bipartite matching algorithm. Graph g should be a
list of neighbors of the left partition, and btoa should be a vector full
of -1's of the same size as the right partition. Returns the size of the
matching. btoa[i] will be the match for vertex i on the right side, or -1
if it's not matched.
Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);
Time: O(√VE)
d93347, 42 lines

bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi&
    B) { //d9e
    if (A[a] != L) return 0;
    A[a] = -1;
    for (int b : g[a]) if (B[b] == L + 1) {
        B[b] = 0;
        if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
            return btoa[b] = a, 1;
    }
    return 0;
} //ad4
int hopcroftKarp(vector<vi>& g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0); //d58
        fill(all(B), 0);
        cur.clear();
        for (int a : btoa) if (a != -1) A[a] = -1;
        rep(a,0,sz(g)) if (A[a] == 0) cur.pb(a);
        for (int lay = 1;; lay++) { //559
            bool islast = 0;
            next.clear();
            for (int a : cur) for (int b : g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay; //1ca
                    islast = 1;
                }
                else if (btoa[b] != a && !B[b]) {
                    B[b] = lay;
                    next.pb(btoa[b]); //c66
                }
            }
            if (islast) break;
            if (next.empty()) return res;
            for (int a : next) A[a] = lay; //4f3
            cur.swap(next);
        }
        rep(a,0,sz(g))
            res += dfs(a, 0, g, btoa, A, B);
    } //67c
}
```

```
DFSMatching.h
Description: Simple bipartite matching algorithm. Graph g should be
a list of neighbors of the left partition, and btoa should be a vector full
of -1's of the same size as the right partition. Returns the size of the
matching. btoa[i] will be the match for vertex i on the right side, or -1
if it's not matched.
Usage: vi btoa(m, -1); dfsMatching(g, btoa);
Time: O(VE)
522b98, 22 lines

bool find(int j, vector<vi>& g, vi& btoa, vi& vis) { //400
    if (btoa[j] == -1) return 1;
    vis[j] = 1; int di = btoa[j];
    for (int e : g[di])
        if (!vis[e] && find(e, g, btoa, vis)) {
            btoa[e] = di; //a0e
            return 1;
        }
    return 0;
}
```

```

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

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

"DFSMatching.h"62c4ec, 20 lines

```

vi cover(vector<vi>& g, int n, int m) {//60f
    vi match(m, -1);
    int res = dfsMatching(g, match);
    vector<bool> lfound(n, true), seen(m);
    for (int it : match) if (it != -1) lfound[it] = false;
    vi q, cover;//2da
    rep(i,0,n) if (lfound[i]) q.pb(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) {//4ed
            seen[e] = true;
            q.pb(match[e]);
        }
    }
    rep(i,0,n) if (!lfound[i]) cover.pb(i);//a72
    rep(i,0,m) if (seen[i]) cover.pb(n+i);
    assert(sz(cover) == res);
    return cover;
}
```

WeightedMatching.h
Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires $N \leq M$.
Time: $\mathcal{O}(N^2M)$

df0677, 31 lines

```

pair<int, vi> hungarian(const vector<vi> &a) {//64f
    if (a.empty()) return {0, {}};
    int n = sz(a) + 1, m = sz(a[0]) + 1;
    vi u(n), v(m), p(m), ans(n - 1);
    rep(i,1,n) {
        p[0] = i;//0b5
        int j0 = 0; // add "dummy" worker 0
        vi dist(m, INT_MAX), pre(m, -1);
        vector<bool> done(m + 1);
        do { // dijkstra
            done[j0] = true;//bd1
            int i0 = p[j0], j1 = -1, delta = INT_MAX;
            rep(j,1,m) if (!done[j]) {
                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                if (dist[j] < delta) delta = dist[j], j1 = j;//865
            }
            rep(j,0,m) {
                if (done[j]) u[p[j]] += delta, v[j] -= delta;
                else dist[j] -= delta;
            }//aa1
        }
    }
}
```

```

        j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
        int j1 = pre[j0];
        p[j0] = p[j1], j0 = j1;//88f
    }
}
rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
return {-v[0], ans}; // min cost
}//cbb
```

GeneralMatching.h
Description: Matching for general graphs. Fails with probability N/mod .
Time: $\mathcal{O}(N^3)$

"../numerical/MatrixInverse-mod.h"1e40dd, 40 lines

```

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

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

    if (M != N) do {//f88
        mat.resize(M, vector<ll>(M));
        rep(i,0,N) {
            mat[i].resize(M);
            rep(j,N,M) {
                int rr = rand() % mod;//b47
                mat[i][j] = rr, mat[j][i] = (mod - rr) % mod;
            }
        }
    } while (matInv(A = mat) != M);
//92b
    vi has(M, 1); vector<pii> ret;
    rep(it,0,M/2) {
        rep(i,0,M) if (has[i])
            rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
                fi = i; fj = j; goto done;//e0a
            }
        } assert(0); done:
        if (fj < N) ret.emplace_back(fi, fj);
        has[fi] = has[fj] = 0;
        rep(sw,0,2) {
            ll a = modpow(A[fi][fj], mod-2);//b7f
            rep(i,0,M) if (has[i] && A[i][fj]) {
                ll b = A[i][fj] * a % mod;
                rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
            }
            swap(fi,fj);//3c7
        }
    }
    return ret;
}
```

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

Time: $\mathcal{O}(E + V)$ c16b3c, 24 lines

```

template<class F> vi scc(const vector<vi> &adj, F f) {//496
    int n = sz(adj);
    vi val(n), comp(n, -1), z, cont;
    int time = 0, ncomps = 0;
    auto dfs = [&](auto &&self, int u) -> int {
        int low = val[u] = ++time, x; z.push_back(u);//ad5
        for (auto e : adj[u]) if (comp[e] < 0)
            low = min(low, val[e] ?: self(self, e));
        if (low == val[u]) {
            do {
                x = z.back(); z.pop_back();//4f1
                comp[x] = ncomps;
                cont.push_back(x);
            } while (x != u);
            f(cont); cont.clear();
            ncomps++;//cda
        }
        return val[u] = low;
    };
    rep(i, 0, n) {
        if (comp[i] < 0) dfs(dfs, i);//418
    }
    return comp;
}
```

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

389e66, 28 lines

```

template<class F>//c2c
void bicomps(vector<vector<pii>> &ed, F f) {
    vi num(sz(ed)), st;
    int t=0;
    auto dfs = [&](auto &&self, int at, int par) -> int {
        int me = num[at] = ++t, top = me;//b12
        for (auto [y, e] : ed[at]) if (e != par) {
            if (num[y]) {
                top = min(top, num[y]);
                if (num[y] < me)
                    st.pb(e);//6e9
            } else {
                int si = sz(st);
                int up = self(self, y, e);
                top = min(top, up);
                if (up == me) {//2cf
                    st.pb(e);
                    f(vi(st.begin() + si, st.end()));
                    st.resize(si);
                }
            }
            else if (up < me) st.pb(e);//51c
            else { /* e is a bridge */ }
        }
    }
    return top;
};//835
rep(i,0,sz(ed)) if (!num[i]) dfs(dfs, i, -1);
}
```

Articulation.h

Description: Finds articulation points (removal separates graph)
Time: $\mathcal{O}(n + m)$

a7b0ba, 25 lines

```
vector<bool> cutpoints(const vector<vi> &adj) {//259
    int timer=0, n=sz(adj);
    vi tin(n,-1), low(n,-1);
    vector<bool> vis(n);
    vector<bool> iscut(n);
    auto dfs = [&](auto &&self, int v, int p) -> void {//7a1
        vis[v] = true;
        tin[v] = low[v] = timer++;
        int ch = 0;
        for (int to : adj[v]) {
            if (to == p) continue;//b9d
            if (vis[to])
                low[v] = min(low[v], tin[to]);
            else {
                self(self, to, v);
                low[v] = min(low[v], low[to]);//914
                if (low[to] >= tin[v] && p!=-1) iscut[v]=1;
                ++ch;
            }
        }
        if(p == -1 && ch > 1) iscut[v]=1;//4e2
    };
    rep(i,0,n) if (!vis[i]) dfs(dfs, i, -1);
    return iscut;
}
```

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.

35fbf7, 56 lines

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

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

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

    void either(int f, int j) {
        f = max(2*f, -1-2*f);
        j = max(2*j, -1-2*j);//2d3
        gr[f].pb(j^1);
        gr[j].pb(f^1);
    }
    void setValue(int x) { either(x, x); }
//41c
    void atMostOne(const vi& li) { // (optional)
        if (sz(li) <= 1) return;
        int cur = ~li[0];
        rep(i,2,sz(li)) {
            int next = addVar();//f5e
            either(cur, ~li[i]);
        }
    }
}
```

```
        either(cur, next);
        either(~li[i], next);
        cur = ~next;
    }//276
    either(cur, ~li[1]);
}

vi val, comp, z; int time = 0;
int dfs(int i) {//1e9
    int low = val[i] = ++time, x; z.pb(i);
    for(int e : gr[i]) if (!comp[e])
        low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
        x = z.back(); z.pop_back();//0c0
        comp[x] = low;
        if (values[x]>>1) == -1)
            values[x]>>1 = x&1;
        } while (x != i);
    return val[i] = low;//749
}

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

3e0eb1, 15 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 nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.
Time: $\mathcal{O}(V + E)$

3e0eb1, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src
    =0) {//fda
    int n = sz(gr);
    vi D(n), its(n), eu(nedges), ret, s = {src};
    D[src]++; // to allow Euler paths, not just cycles
    while (!s.empty()) {
        int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
        //39a
        if (it == end){ ret.pb(x); s.pop_back(); continue; }
        tie(y, e) = gr[x][it++];
        if (!eu[e]) {
            D[x]--, D[y]++;
            eu[e] = 1; s.pb(y);//f91
        }
    }
    for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
    return {ret.rbegin(), ret.rend()};
}
```

7.4 Coloring

EdgeColoring.h

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

ca07a0, 31 lines

```
vi edgeColoring(int N, vector<pii> eds) {//d26
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
```

```
    for (pii e : eds) {//945
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1) //665
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd
            ])
            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) {//f7e
            int left = fan[i], right = fan[++i], x = cc[i];
            adj[u][x] = left;
            adj[left][x] = u;
            adj[right][x] = -1;
            free[right] = x;//e59
        }
        adj[u][d] = fan[i];
        adj[fan[i]][d] = u;
        for (int y : {fan[0], u, end})
            for (int& z = free[y] = 0; adj[y][z] != -1; z++);//b06
    }
    rep(i,0,sz(eds))
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i
            ];
    return ret;
}//cbb
```

b0d5b1, 12 lines

7.5 Heuristics

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.
Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

b0d5b1, 12 lines

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

MaximumClique.h

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.
Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

108bd4, 49 lines

```
typedef vector<bitset<200>> vb;//b92
struct Maxclique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; };
    typedef vector<Vertex> vv;
    vb e;//5b2
    vv V;
    vector<vi> C;
    vi qmax, q, S, old;
    void init(vv& r) {
        for (auto& v : r) v.d = 0;//dab
    }
}
```

```

    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
    sort(all(r), [](auto a, auto b) { return a.d > b.d; });
    int mxD = r[0].d;
    rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
} //a6a
void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
        if (sz(q) + R.back().d <= sz(qmax)) return; //62e
        q.pb(R.back().i);
        vv T;
        for (auto v:R) if (e[R.back().i][v.i]) T.pb({v.i});
        if (sz(T)) {
            if (S[lev]++ / ++pk < limit) init(T); //feb
            int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
            C[1].clear(), C[2].clear();
            for (auto v : T) {
                int k = 1;
                auto f = [&](int i) { return e[v.i][i]; }; //94f
                while (any_of(all(C[k]), f)) k++;
                if (k > mxk) mxk = k, C[mxk + 1].clear();
                if (k < mnk) T[j++] .i = v.i;
                C[k].pb(v.i);
            } //08b
            if (j > 0) T[j - 1].d = 0;
            rep(k,mnk,mxk + 1) for (int i : C[k])
                T[j].i = i, T[j++].d = k;
            expand(T, lev + 1);
        } else if (sz(q) > sz(qmax)) qmax = q; //15f
        q.pop_back(), R.pop_back();
    }
}
vi maxClique() { init(V), expand(V); return qmax; }
MaxClique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S)
) {} //83c
rep(i,0,sz(e)) V.pb({i});
};

```

7.6 Trees

CompressTree.h

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

"LCA.h" cea406, 21 lines

```

typedef vector<pair<int, int>> vpi; //386
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.time));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp); //3b2
    int m = sz(li) - 1;
    rep(i,0,m) {
        int a = li[i], b = li[i+1];
        li.pb(lca.lca(a, b));
    } //c76
    sort(all(li), cmp);
    li.erase(unique(all(li)), li.end());
    rep(i,0,sz(li)) rev[li[i]] = i;
    vpi ret = {pii(0, li[0])};
    rep(i,0,sz(li)-1) {} //ff8
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.lca(a, b)], b);
    }
    return ret;
} //cbb

```

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most $\log(n)$ light edges. Takes as input the full adjacency list. op edges being true means that values are stored in the edges, as opposed to the nodes.
Time: $\mathcal{O}(\log N)$

f6f4ce, 55 lines

```

template<bool op_edges = false> //1ab
struct hld {
    vector<vi> adj;
    vi par, size, in, head, d;
    hld(int n) : adj(n), par(n), size(n), in(n), head(n), d(n) {}
} //c22
void add_edge(int u, int v) {
    adj[u].pb(v);
    adj[v].pb(u);
}
//9ee
void dfs_size(int v = 0, int p = 0) {
    size[v] = 1;
    for (int &e : adj[v]) {
        if (e != p) {
            d[e] = d[v] + 1; //a74
            par[e] = v;
            dfs_size(e, v);
            size[v] += size[e];
            if (size[e] > size[adj[v][0]] || adj[v][0] == p)
                swap(e, adj[v][0]);
        } //96d
    }
}

void dfs_hld(int v = 0, int p = 0) {
    static int t = 0; //978
    in[v] = t++;
    for (int e : adj[v]) {
        if (e != p) {
            if (e == adj[v][0]) {
                head[e] = head[v]; //c3e
            } else {
                head[e] = e;
            }
            dfs_hld(e, v);
        } //47a
    }
}

```

```

template<typename F>
void op_path(int x, int y, F op) {} //b81
while (head[x] != head[y]) {
    if (d[head[x]] > d[head[y]]) swap(x, y);
    op(in[head[y]], in[y] + 1);
    y = par[head[y]];
} //387
if (d[x] > d[y]) swap(x, y);
op(in[x] + (op_edges ? 1 : 0), in[y] + 1);
}

```

```

template<typename F> //d25
void op_subtree(int x, F op) {
    op(in[x] + (op_edges ? 1 : 0), in[x] + size[x]);
}
};

```

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

0ff462, 90 lines

```

struct Node { // Splay tree. Root's pp contains tree's
    parent; //a4e
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this; //b8f
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
    void pushFlip() {
        if (!flip) return; //dfd
        flip = 0; swap(c[0], c[1]);
        if (c[0]) c[0]->flip ^= 1;
        if (c[1]) c[1]->flip ^= 1;
    }
    int up() { return p ? p->c[1] == this : -1; } //3a9
    void rot(int i, int b) {
        int h = i ^ b;
        Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
        if ((y->p == p)) p->c[up[i]] = y;
        c[i] = z->c[i ^ 1]; //eb7
        if (b < 2) {
            x->c[h] = y->c[h ^ 1];
            y->c[h ^ 1] = x;
        }
        z->c[i ^ 1] = this; //430
        fix(); x->fix(); y->fix();
        if (p) p->fix();
        swap(pp, y->pp);
    }
    void splay() {} //4c8
    for (pushFlip(); p; ) {
        if (p->p) p->p->pushFlip();
        p->pushFlip(); pushFlip();
        int c1 = up(), c2 = p->up();
        if (c2 == -1) p->rot(c1, 2); //9e8
        else p->p->rot(c2, c1 != c2);
    }
}
Node* first() {
    pushFlip(); //828
    return c[0] ? c[0]->first() : (splay(), this);
}
};

struct LinkCut {} //d99
vector<Node> node;
LinkCut(int N) : node(N) {}

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

```

```

void makeRoot(Node* u) {
    access(u);
    u->splay(); //09d
    if(u->c[0]) {
        u->c[0]->p = 0;
        u->c[0]->flip ^= 1;
        u->c[0]->pp = u;
        u->c[0] = 0; //41e
        u->fix();
    }
}
Node* access(Node* u) {
    u->splay(); //4e7
    while (Node* pp = u->pp) {
        pp->splay(); u->pp = 0;
        if (pp->c[1]) {
            pp->c[1]->p = 0; pp->c[1]->pp = pp; }
        pp->c[1] = u; pp->fix(); u = pp; //f4d
    }
    return u;
}
};

```

DirectedMST.h

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

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

"../data-structures/UnionFindRollback.h" 057d96, 60 lines

```

struct Edge { int a, b; ll w{}; }; //4d9
struct Node {
    Edge key;
    Node *l=0, *r=0;
    ll delta{};
    void prop() { //936
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    } //5dc
    Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? b;
    a->prop(), b->prop(); //72a
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); } //8e9

```

```

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

```

```

        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.join(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cyscs.push_front({u, time, {&Q[qi], &Q[end]}}); //984
    }
    rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
}
//eba
for (auto& [u,t,cc] : cyscs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : cc) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge; //ffd
}
rep(i,0,n) par[i] = in[i].a;
return {res, par};
}

```

LCA.h

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

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

"../data-structures/RMQ.h" aa4761, 21 lines

```

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

    LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret)) {} //1e9
    void dfs(vector<vi>& C, int v, int par) {
        time[v] = T++;
        for (int y : C[v]) if (y != par) {
            path.pb(v), ret.push_back(time[v]);
            dfs(C, y, v); //3f8
        }
    }

    int lca(int a, int b) {
        if (a == b) return a; //3f5
        tie(a, b) = minmax(time[a], time[b]);
        return path[rmq.query(a, b)];
    }
    //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
}; //214

```

7.7 Math

7.7.1 Number of Spanning Trees

Create an $N \times N$ matrix 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).

7.7.2 Erdős–Gallai theorem

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

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

Geometry (8)

8.1 Geometric primitives

Point.h

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

634da7, 29 lines

```

template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
//fa7
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T _x=0, T _y=0) : x(_x), y(_y) {} //a5f
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    P operator*(T d) const { return P(x*d, y*d); } //e11
    P operator/(T d) const { return P(x/d, y/d); }
    T dot(P p) const { return x*p.x + y*p.y; }
    T cross(P p) const { return x*p.y - y*p.x; } // + => p on right
    T cross(P a, P b) const { return (a-*this).cross(b-*this); }
    bool half() const { return y < 0 || (y == 0 && x < 0); } //053
    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//8da
    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)); //ad4
    }
    friend ostream& operator<<(ostream& os, P p) {
        return os << "(" << p.x << ", " << p.y << ")"; }
};

```

AngleSort.h

Description: Sorts points radially across the origin. To sort around a point, sort a-p and b-p.

"Point.h" c10d46, 7 lines

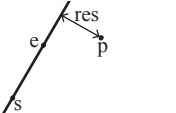
```

template<class P> //159
void anglesort(vector<P> &v, P p=P(0, 0)) {
    sort(all(v), [p](P a, P b) {
        a = a - p, b = b - p;
        return a.half() == b.half() ? a.cross(b) > 0 : a.half() < b.half();
    }); //b97
}

```

lineDistance.h

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

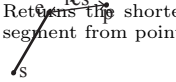


"Point.h" f6bf6b, 4 lines

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

SegmentDistance.h

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



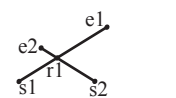
"Point.h" 5c88f4, 6 lines

```
typedef Point<double> P; //b95
double segDist(P& s, P& e, P& p) {
    if (s==e) return (p-s).dist();
    auto d = (e-s).dist2(), t = min(d,max(.0, (p-s).dot(e-s)))
        ;
    return ((p-s)*d-(e-s)*t).dist()/d;
} //cbb
```

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

SegmentIntersection.h


Description:
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
    //dec
    auto oa = c.cross(d, a), ob = c.cross(d, b),
        oc = a.cross(b, c), od = a.cross(b, d);
    // Checks if intersection is single non-endpoint point.
    if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
        return {(a * ob - b * oa) / (ob - oa)}; //8a0
    set<P> s;
    if (onSegment(c, d, a)) s.insert(a);
    if (onSegment(c, d, b)) s.insert(b);
    if (onSegment(a, b, c)) s.insert(c);
    if (onSegment(a, b, d)) s.insert(d); //814
    return {all(s)};
}
```

Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;



"Point.h" a01f81, 8 lines

```
template<class P> //47e
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
    auto d = (e1 - s1).cross(e2 - s2);
    if (d == 0) // if parallel
        return {-(s1.cross(e1, s2) == 0), P(0, 0)};
    auto p = s2.cross(e1, e2), q = s2.cross(e2, s1); //16d
    return {1, (s1 * p + e1 * q) / d};
}
```

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

sideOf.h

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

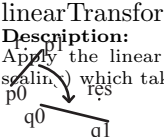
Usage: bool left = sideOf(p1,p2,q)==1;

OnSegment.h

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

linearTransformation.h

Description:
Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.



"Point.h" 03a306, 6 lines

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

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

LineProjectionReflection.h

Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab instead. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

CircleIntersection.h

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

Usage: 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
; //347
if (sum*sum < d2 || dif*dif > d2) return false;
P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
*out = {mid - per, mid + per};
return true;
} //cbb

CircleTangents.h

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

CircleLine.h

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>//64a
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}; //fd3
    P h = ab.unit() * sqrt(h2);
    return {p - h, p + h};
}
```

CirclePolygonIntersection.h

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

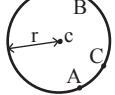
Time: $\mathcal{O}(n)$

```
"../content/geometry/Point.h" ale63, 19 lines
typedef Point<double> P; //a6c
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p; //eda
        auto a = d.dot(p) / d.dist2(), b = (p.dist2() - r*r) / d.dist2();
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
        if (t < 0 || 1 <= s) return arg(p, q) * r2; //174
        P u = p + d * s, v = p + d * t;
        return arg(p, u) * r2 + u.cross(v) / 2 + arg(v, q) * r2;
    };
    auto sum = 0.0;
    rep(i, 0, sz(ps)) //a61
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
    return sum;
}
```

circumcircle.h

Description:

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



```
"Point.h" 1caa3a, 9 lines
typedef Point<double> P; //032
double ccRadius(const P& A, const P& B, const P& C) {
    return (B-A).dist() * (C-B).dist() * (A-C).dist() /
        abs((B-A).cross(C-A)) / 2;
}
P ccCenter(const P& A, const P& B, const P& C) { //793
    P b = C-A, c = B-A;
    return A + (b*c.dist2() - c*b.dist2()).perp() / b.cross(c) / 2;
}
```

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.

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

```
"circumcircle.h" 09dd0a, 17 lines
pair<P, double> mec(vector<P> ps) { //b50
    shuffle(all(ps), mt19937(time(0)));
}
```

```
P o = ps[0];
double r = 0, EPS = 1 + 1e-8;
rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0; //d54
    rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
        o = (ps[i] + ps[j]) / 2;
        r = (o - ps[i]).dist();
        rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS) {
            o = ccCenter(ps[i], ps[j], ps[k]); //4ec
            r = (o - ps[i]).dist();
        }
    }
}
return {o, r}; //2ac
}
```

8.3 Polygons

InsidePolygon.h

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

Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};

bool in = inPolygon(v, P{3, 3}, false);

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

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

PolygonArea.h

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

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

PolygonCenter.h

Description: Returns the center of mass for a polygon.

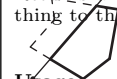
Time: $\mathcal{O}(n)$

```
"Point.h" 9706dc, 9 lines
typedef Point<double> P; //082
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]); //168
    }
    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" 056a39, 13 lines
typedef Point<double> P; //366
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; //c08
        if (side != (s.cross(e, prev) < 0))
            res.pb(lineInter(s, e, cur, prev).second);
        if (side)
            res.pb(cur);
    } //0e1
    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; //49c
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])]; //896
        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); //407
                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) { //8be
                    segs.emplace_back(rat(C - A, B - A), 1);
                    segs.emplace_back(rat(D - A, B - A), -1);
                }
            }
        } //155
    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)) { //88e
        if (!cnt) sum += segs[j].first - segs[j - 1].first;
        cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
} //f48
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: $O(n \log n)$

```
"Point.h"
310954, 13 lines

typedef Point<ll> P; //3e3
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; //f18
    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;
        } //aa0
    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: $O(n)$

```
"Point.h"
c571b8, 12 lines

typedef Point<ll> P; //5c7
array<P, 2> hullDiameter(vector<P> S) {
    int n = sz(S), j = n < 2 ? 0 : 1;
    pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
    rep(i, 0, j)
        for (; j = (j + 1) % n) { //56c
            res = max(res, {{S[i] - S[j]}.dist2(), {S[i], S[j]}})
                ;
            if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >=
                0)
                break;
        }
    return res.second; //52a
}
```

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: $O(\log N)$

```
"Point.h", "sideOf.h", "OnSegment.h"
71446b, 14 lines

typedef Point<ll> P; //7a3

bool inHull(const vector<P>& l, P p, bool strict = true) {
    int a = 1, b = sz(l) - 1, r = !strict;
    if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
    if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b); //4a6
    if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p) <=
        -r)
        return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        (sideOf(l[0], l[c], p) > 0 ? b : a) = c; //0da
    }
    return sgn(l[a].cross(l[b], p)) < r;
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: • $(-1, -1)$ if no collision, • $(i, -1)$ if touching the corner i , • (i, i) if along side $(i, i + 1)$, • (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: $O(\log n)$

```
"Point.h"
7cf45b, 39 lines

#define cmp(i, j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%
    n])) //b9d
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) { //51a
        int m = (lo + hi) / 2;
        if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) =
            m;
    } //e8c
    return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P> //7fd
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
    int endA = extrVertex(poly, (a - b).perp());
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpL(endA) < 0 || cmpL(endB) > 0)
        return {-1, -1}; //04b
    array<int, 2> res;
    rep(i, 0, 2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n; //ec0
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpL(hi)) % n;
        swap(endA, endB);
    } //6ab
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpL(res[0]) && !cmpL(res[1]))
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]}; //08a
        }
    return res;
}
```

HullTangents.h

Description: Finds the two tangent vertices on the convex hull to some point. Point must be outside. Appears to be left then right.

```
"Point.h"
0bdcf, 22 lines

template<typename P, typename F> //134
int extremeVertex(const P& poly, F direction) {
    int n = sz(poly), l = 0, ls;
    auto vertexCmp = [&](int i, int j) {
        return sgn(direction(poly[j]).cross(poly[j] - poly[i]));
    };
    auto isExtreme = [&](int i, int& is) { //d3d
        return (is = vertexCmp((i+1)%n, i)) >= 0 && vertexCmp(i,
            (i+n-1)%n) < 0; };
    for (int r = isExtreme(0, ls) ? 1 : n; l + 1 < r;) {
        int m = (l + r) / 2, ms;
        if (isExtreme(m, ms)) return m;
    }
}
```

```
if (ls != ms ? ls < ms : ls == vertexCmp(l, m)) r = m; //
    beb
else l = m, ls = ms;
}
return l;
}
//d22
template<typename P>
pair<int, int> tangentsConvex(const P &point, const vector<
    P>& poly) {
    return {
        extremeVertex(poly, [&](const P& q) { return q - point; }
            ),
        extremeVertex(poly, [&](const P& q) { return point - q; }
            )}; //fa7
}
```

MinkowskiSum.h

Description: Minkowski sum of set of convex ccw polygons.

Time: $O(P \log N)$, where P is number of points and N is number of polygons.

```
"Point.h"
ecfe9a, 33 lines

typedef Point<ll> P; //657
vector<P> minkowskiSum(vector<vector<P>>> hs) {
    auto cmp = [](P a, P b) {
        return make_pair(a.x < 0 || a.x == 0 && a.y < 0, a.y *
            (ll)b.x)
            < make_pair(b.x < 0 || b.x == 0 && b.y < 0, a.x * (ll
                )b.y);
    }; //289
    typedef tuple<P, int, int> T;
    auto cmp_tup = [&cmp](T a, T b) {
        auto& [pa, ja, ia] = a;
        auto& [pb, jb, ib] = b;
        if (cmp(pa, pb)) return false; //9d0
        if (cmp(pb, pa)) return true;
        return make_pair(ja, ia) < make_pair(jb, ib);
    };
    priority_queue<T, vector<T>, decltype(cmp_tup)> pq(
        cmp_tup);
    P cur = P(); //404
    int s = 0, t = 0;
    rep(i, 0, sz(hs)) {
        auto& v = hs[i];
        rotate(begin(v), min_element(all(v)), end(v));
        if (sz(v) > 1) pq.push({v[1] - v[0], 0, i}), s += sz(v)
            ; //cee
        cur = cur + v[0];
    }
    vector<P> h(s + 1);
    for (h[t++] = cur; sz(pq);) {
        auto [p, j, i] = pq.top(); pq.pop(); //451
        t -= (t >= 2 && !cmp(h[t - 1] - h[t - 2], p));
        h[t++] = (cur = cur + p);
        auto& v = hs[i];
        if (++j < sz(v)) pq.push({v[(j + 1) % sz(v)] - v[j], j,
            i});
    } //297
    return {h.begin(), h.begin() + t - (t >= 2 && h[0] == h[t
        - 1])};
}
```

8.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.

Time: $O(n \log n)$

```
"Point.h"
ac41a6, 17 lines

typedef Point<ll> P; //9e7
pair<P, P> closest(vector<P> v) {
    assert(sz(v) > 1);
}
```

```
set<P> S;
sort(all(v), [](P a, P b) { return a.y < b.y; });
pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}}; //e83
int j = 0;
for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d); //cb2
    for (; lo != hi; ++lo)
        ret = min(ret, {(p - p).dist2(), {p, p}});
    S.insert(p);
}
return ret.second; //982
}
```

ManhattanMST.h

Description: Given N points, returns up to $4N$ 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"	e2611c, 23 lines
<pre>typedef Point<int> P; //bde vector<array<int, 3>> manhattanMST(vector<P> ps) { vi id(sz(ps)); iota(all(id), 0); vector<array<int, 3>> edges; rep(k, 0, 4) { //9bd sort(all(id), [&](int i, int j) { return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y; }); map<int, int> sweep; for (int i : id) { for (auto it = sweep.lower_bound(-ps[i].y); //0bb it != sweep.end(); sweep.erase(it++)) { int j = it->second; P d = ps[i] - ps[j]; if (d.y > d.x) break; edges.pb({d.y + d.x, i, j}); //868 } sweep[-ps[i].y] = i; } for (P& p : ps) if ((k & 1) * p.x == -p.x; else swap(p.x, p.y)); } //aa4 return edges; }</pre>	

kdTree.h

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

"Point.h"	bac5b0, 63 lines
<pre>typedef long long T; //632 typedef Point<T> P; const T INF = numeric_limits<T>::max(); bool on_x(const P& a, const P& b) { return a.x < b.x; } bool on_y(const P& a, const P& b) { return a.y < b.y; } //c56</pre>	

```
bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; } //c56
```

```
struct Node {
    P pt; // if this is a leaf, the single point in it
    T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
    Node *first = 0, *second = 0; //5b4
```

```
T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x, y) - p).dist2(); //a82
```

```
}

Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
        x0 = min(x0, p.x); x1 = max(x1, p.x); //151
        y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
        // split on x if width >= height (not ideal...)
        sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y); //1d2
        // divide by taking half the array for each child (not
        // best performance with many duplicates in the middle)
        int half = sz(vp)/2;
        first = new Node({vp.begin(), vp.begin() + half});
        second = new Node({vp.begin() + half, vp.end()}); //ace
    }
}

};

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

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

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

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

FastDelaunay.h

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

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

"Point.h"	04ae3a, 88 lines
<pre>typedef Point<ll> P; //503 typedef struct Quad* Q; typedef __int128_t lll; // (can be ll if coords are < 2e4) P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point struct Quad { //8bb Q rot, o; P p = arb; bool mark; P& F() { return r()->p; } Q& r() { return rot->rot; } Q prev() { return rot->o->rot; }</pre>	

```
Q next() { return r()->prev(); } //0bd
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle ?
    ll p2 = p.dist2(), A = a.dist2()-p2,
        B = b.dist2()-p2, C = c.dist2()-p2; //520
    return p.cross(a, b)*C + p.cross(b, c)*A + p.cross(c, a)*B > 0;
}

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

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

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

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

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

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

```
vector<P> triangulate(vector<P> pts) { //e5d
    sort(all(pts)); assert(unique(all(pts)) == pts.end());
```

```
    if (sz(pts) < 2) return {};\n    Q e = rec(pts).first;\n    vector<Q> q = {e};\n    int qi = 0; //dd4\n    while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;\n#define ADD { Q c = e; do { c->mark = 1; pts.pb(c->p); \\\n    q.pb(c->r()); c = c->next(); } while (c != e); }\n    ADD; pts.clear();\n    while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD; //24a\n    return pts;\n}\n
```

8.5 3D PolyhedronVolume.h

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

```
3058c3, 6 lines\n\ntemplate<class V, class L> //27c\ndouble signedPolyVolume(const V& p, const L& trilst) {\n    double v = 0;\n    for (auto i : trilst) v += p[i.a].cross(p[i.b]).dot(p[i.c]);\n    return v / 6;\n} //cbb\n
```

Point3D.h

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

```
6eb43e, 32 lines\n\ntemplate<class T> struct Point3D { //811\n    typedef Point3D P;\n    typedef const P& R;\n    T x, y, z;\n    explicit Point3D(T _x=0, T _y=0, T _z=0) : x(_x), y(_y),\n        z(_z) {\n    bool operator<(R p) const { //5e8\n        return tie(x, y, z) < tie(p.x, p.y, p.z); }\n    bool operator==(R p) const {\n        return tie(x, y, z) == tie(p.x, p.y, p.z); }\n    P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }\n    P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }\n    //9b1\n    P operator*(T d) const { return P(x*d, y*d, z*d); }\n    P operator/(T d) const { return P(x/d, y/d, z/d); }\n    T dot(R p) const { return x*p.x + y*p.y + z*p.z; }\n    P cross(R p) const {\n        return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);\n    } //58a\n    T dist2() const { return x*x + y*y + z*z; }\n    double dist() const { return sqrt((double)dist2()); }\n    //Azimuthal angle (longitude) to x-axis in interval [-pi,\n    pi]\n    double phi() const { return atan2(y, x); } //a2c\n    //Zenith angle (latitude) to the z-axis in interval [0,\n    pi]\n    double theta() const { return atan2(sqrt(x*x+y*y),z); }\n    P unit() const { return *this/(T)dist(); } //makes dist() =1\n    //returns unit vector normal to *this and p\n    P normal(P p) const { return cross(p).unit(); } //e88\n    //returns point rotated 'angle' radians ccw around axis\n    P rotate(double angle, P axis) const {\n        double s = sin(angle), c = cos(angle); P u = axis.unit\n        ();\n        return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;\n    } //e03\n};\n
```

3dHull.h

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

```
Time: O(n^2)\n\"Point3D.h\" ce1872, 49 lines\n\ntypedef Point3D<double> P3; //e28\n\nstruct PR {\n    void ins(int x) { (a == -1 ? a : b) = x; }\n    void rem(int x) { (a == x ? a : b) = -1; }\n    int cnt() { return (a != -1) + (b != -1); } //c34\n    int a, b;\n};\n\nstruct F { P3 q; int a, b, c; };\n//36b\nvector<F> hull3d(const vector<P3>& A) {\n    assert(sz(A) >= 4);\n    vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));\n#define E(x,y) E[f.x][f.y]\n    vector<F> FS; //de0\n    auto mf = [&](int i, int j, int k, int l) {\n        P3 q = (A[j] - A[i]).cross((A[k] - A[i]));\n        if (q.dot(A[l]) > q.dot(A[i]))\n            q = q * -1;\n        F f{q, i, j, k}; //2be\n        E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);\n        FS.pb(f);\n    };\n    rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)\n        mf(i, j, k, 6 - i - j - k); //e21\n\n    rep(i,4,sz(A)) {\n        rep(j,0,sz(FS)) {\n            F f = FS[j];\n            if(f.q.dot(A[i]) > f.q.dot(A[f.a])) { //b63\n                E(a,b).rem(f.c);\n                E(a,c).rem(f.b);\n                E(b,c).rem(f.a);\n                swap(FS[j--], FS.back());\n                FS.pop_back(); //0df\n            }\n        }\n        int nw = sz(FS);\n        rep(j,0,nw) {\n            F f = FS[j]; //945\n#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f\n                .c);\n            C(a, b, c); C(a, c, b); C(b, c, a);\n        }\n        for (F& it : FS) if ((A[it.b] - A[it.a]).cross(//ab3\n            A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);\n        return FS;\n    };\n};\n
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_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.

```
611f07, 8 lines\n\ndouble sphericalDistance(double f1, double t1, //6da\n    double f2, double t2, double radius) {\n    double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);\n    double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);\n
```

```
double dz = cos(t2) - cos(t1);\n    double d = sqrt(dx*dx + dy*dy + dz*dz); //65e\n    return radius*2*asin(d/2);\n}\n
```

SegmentDistance3D.h

Description: returns closest two points from each 3D segment

```
\"P3oint3D.h\" 10f4a2, 40 lines\n\ntypedef Point3D<double> P3; //63e\n//returns closest two points from each 3d segment\npair<P3, P3> segmentDistance3d(P3 s1, P3 e1, P3 s2, P3 e2)\n{\n    pair<P3, P3> res{s1, s2};\n    auto check = [&res](P3 a, P3 b) {\n        if((b-a).dist() < (res.second-res.first).dist()) //4d1\n            res = {a, b};\n    };\n    //check endpoint-endpoint\n    check(s1, s2);\n    check(s1, e2); //9e7\n    check(e1, s2);\n    check(e1, e2);\n\n    P3 d1 = (e1-s1).unit();\n    double t1Max = (e1-s1).dist(); //3e0\n    P3 d2 = (e2-s2).unit();\n    double t2Max = (e2-s2).dist();\n    //check endpoint-segment, dist from p to the line segment\n        s + t*d\n    auto pointLine = [&](P3 p, P3 s, P3 d, double tMax) {\n        P3 v = p-s; //f4b\n        double t = v.dot(d);\n        if(0 <= t && t <= tMax)\n            check(s+d*t, p);\n    };\n    pointLine(s1, s2, d2, t2Max); //ee a\n    pointLine(e1, s2, d2, t2Max);\n    pointLine(s2, s1, d1, t1Max);\n    pointLine(e2, s1, d1, t1Max);\n\n    //check segment-segment //491\n    P3 n = d1.cross(d2);\n    if(n.dist2() != 0) { //only check if not parallel;\n        parallel case is handled by pointLine checks\n        double t1 = (d2.cross(n)).dot(s2-s1)/n.dot(n);\n        double t2 = (d1.cross(n)).dot(s2-s1)/n.dot(n);\n        if(0 <= t1 && t1 <= t1Max && 0 <= t2 && t2 <= t2Max) //25f\n            check(s1+d1*t1, s2+d2*t2);\n    }\n    return res;\n}\n
```

PointToFace.h

Description: Finds the distance between a point and a 3d hull face

```
\"Point3D.h\", \"3dHull.h\" a9d101, 6 lines\n\ntypedef Point3D<double> P3; //f1f\ndouble pointToFace(P3 p, F f) {\n    auto dir = f.q.unit() * -1;\n    auto vec = p - pts[f.a];\n    return vec.dot(dir);\n} //cbb\n
```

To project points onto a plane and map plane to xy -plane, project point onto the plane, use normalized $(B - A)$ and $(C - A)$ as unit vectors, and find projected point as linear combination of basis vectors on xy -plane. $(B - A)$ maps to $(1, 0)$, $(C - A)$ maps to $(\cos \theta, \sin \theta)$, where θ is the angle between $B - A$ and $C - A$.

Strings (9)

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

```
d4f9aa, 16 lines
vi pi(const string& s) { //f6d
    vi p(sz(s));
    rep(i, 1, sz(s)) {
        int g = p[i-1];
        while (g && s[i] != s[g]) g = p[g-1];
        p[i] = g + (s[i] == s[g]); //0ff
    }
    return p;
}
```

```
vi match(const string& s, const string& pat) { //9e6
    vi p = pi(pat + '\0' + s), res;
    rep(i, sz(p)-sz(s), sz(p))
        if (p[i] == sz(pat)) res.pb(i - 2 * sz(pat));
    return res;
} //cbb
```

Zfunc.h
Description: z[i] computes the length of the longest common prefix of s[i] and s, except z[0] = 0. (abacaba -> 0010301)
Time: $\mathcal{O}(n)$

```
ee09e2, 12 lines
vi Z(const string& S) { //fc3
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i, 1, sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - l]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]]) //8ec
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z; //939
}
```

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

```
e7ad79, 13 lines
array<vi, 2> manacher(const string& s) { //510
    int n = sz(s);
    array<vi, 2> p = {vi(n+1), vi(n)};
    rep(z, 0, 2) for (int i=0, l=0, r=0; i < n; i++) {
        int t = r-i+1z;
        if (i<r) p[z][i] = min(t, p[z][l+t]); //f50
        int L = i-p[z][i], R = i+p[z][i]-!z;
        while (L>=1 && R+1<n && s[L-1] == s[R+1])
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    } //291
    return p;
}
```

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

```
d07a42, 8 lines
int minRotation(string s) { //20f
    int a=0, N=sz(s); s += s;
    rep(b, 0, N) rep(k, 0, N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1);
            break;}
        if (s[a+k] > s[b+k]) { a = b; break; }
    } //3a8
    return a;
}
```

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

```
bc716b, 22 lines
struct SuffixArray { //7a7
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)), y(n), ws(max(n, lim));
        x.push_back(0), sa = lcp = y, iota(all(sa), 0); //7c9
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i, 0, n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i, 0, n) ws[x[i]]++; //f08
            rep(i, 1, lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[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++; //726
        }
        for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
            for (k && k--, j = sa[x[i] - 1];
                s[i + k] == s[j + k]; k++);
    } //e03
};
```

Hashing.h
Description: Self-explanatory methods for string hashing.

```
e5966ff, 44 lines
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
//d41
// code, but works on evil test data (e.g. Thue-Morse,
// where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is
// random,
// or work mod 10^9+7 if the Birthday paradox is not a
// problem.
typedef uint64_t ull; //98c
struct H {
    ull x; H(ull x=0) : x(x) {}
    H operator+(H o) { return x + o.x + (x + o.x < x); }
    H operator-(H o) { return *this + ~o.x; }
    H operator*(H o) { auto m = (__uint128_t)x * o.x; //884
        return H((ull)m) + (ull)(m >> 64); }
    ull get() const { return x + !~x; }
```

```
bool operator==(H o) const { return get() == o.get(); }
bool operator<(H o) const { return get() < o.get(); }
}; //7dd
static const H C = (1ll)1e11+3; // (order ~ 3e9; random also
ok)
```

```
struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string& str) : ha(sz(str)+1), pw(ha) { //c1e
        pw[0] = 1;
        rep(i, 0, sz(str))
            ha[i+1] = ha[i] * C + str[i],
            pw[i+1] = pw[i] * C;
    } //b8f
    H hashInterval(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
};
//4b7
vector<H> getHashes(string& str, int length) {
    if (sz(str) < length) return {};
    H h = 0, pw = 1;
    rep(i, 0, length)
        h = h * C + str[i], pw = pw * C; //6b3
    vector<H> ret = {h};
    rep(i, length, sz(str)) {
        ret.pb(h = h * C + str[i] - pw * str[i-length]);
    }
    return ret; //413
}
```

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

AhoCorasick.h
Description: Constructs Aho-Corasick automaton for given list of words
Time: Construction is $\mathcal{O}(N)$, move() is amortized $\mathcal{O}(N)$

```
2d48ca, 34 lines
struct AhoCorasick { //be4
    struct Node {
        map<char, int> nxt;
        int lnk = 0, cnt = 0;
    };
    vector<Node> t; //b46
    AhoCorasick(vector<string> words) {
        t.pb({});
        for (string s : words) {
            int cur = 0;
            for (char c : s) { //638
                if (!t[cur].nxt[c]) {
                    t[cur].nxt[c] = sz(t);
                    t.pb({});
                }
                cur = t[cur].nxt[c]; //05c
            }
            t[cur].cnt++;
        }
        queue<int> q; q.push(0);
        while (!q.empty()) { //b39
            int u = q.front(); q.pop();
            assert(0 <= u && u < sz(t));
            for (auto &p : t[u].nxt) {
                if (u) t[p.second].lnk = move(t[u].lnk, p.first);
                q.push(p.second); //e67
            }
        }
    }
    int move(int u, int c) {
        if (t[u].nxt[c]) return t[u].nxt[c]; //1f8
        return u == 0 ? 0 : t[u].nxt[c] = move(t[u].lnk, c);
    }
};
```

```
    }
};
```

SuffixAutomaton.h

Description: Constructs a suffix automaton on string s cnt() constructs an array of equivalence class sizes first() constructs an array of first occurrences for each node

Time: All functions are $\mathcal{O}(N)$.

<pre>struct SuffixAutomaton { //3d3 struct Node { int len = 0, lnk = 0; int nxt[26]; }; string s; //f20 vector<Node> t; int last = 0; SuffixAutomaton(string s = "") { t.pb({0, -1, {}}); for (char c : s) add(c); } //a4a void add(char c) { s += c; c -= 'a'; int u = last; int v = last = sz(t); t.pb({t[u].len + 1, 0, {}}); while (u >= 0 && !t[u].nxt[c]) t[u].nxt[c] = v, u = t[u].lnk; //b21 if (u == -1) return; int q = t[u].nxt[c]; if (t[u].len + 1 == t[q].len) { t[v].lnk = q; return; } int cpy = sz(t); t.pb(t[q]); //aa4 t[cpy].len = t[u].len + 1; while (u >= 0 && t[u].nxt[c] == q) t[u].nxt[c] = cpy, u = t[u].lnk; t[v].lnk = t[q].lnk = cpy; } //2f2 vector<int> cnt() { vector<int> res(sz(t), 0); int cur = 0; for (char c : s) res[cur = t[cur].nxt[c - 'a']]++; //82c vector<pair<int, int>> srt; for (int i = 1; i < sz(t); i++) srt.pb({-t[i].len, i}); sort(all(srt)); for (auto &p : srt) //cc8 res[t[p.second].lnk] += res[p.second]; return res; } vector<int> first() { vector<int> res(sz(t), sz(s)); //c3b int cur = 0; for (int i = 0; i < sz(s); i++) { cur = t[cur].nxt[s[i] - 'a']; res[cur] = min(res[cur], i); } //268 vector<pair<int, int>> srt; for (int i = 1; i < sz(t); i++) srt.pb({-t[i].len, i}); sort(all(srt)); for (auto &p : srt) //23d res[t[p.second].lnk] = min(res[t[p.second].lnk], res[p.second]); return res; } };</pre>	24974f, 55 lines
--	------------------

<h2>Various (10)</h2> <h3>10.1 Intervals</h3> <p>IntervalContainer.h</p> <p>Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive].</p> <p>Time: $\mathcal{O}(\log N)$</p>	edce47, 23 lines
<pre>set<pii>::iterator addInterval(set<pii>& is, int L, int R) { //ba1 if (L == R) return is.end(); auto it = is.lower_bound({L, R}), before = it; while (it != is.end() && it->first <= R) { R = max(R, it->second); before = it = is.erase(it); //ea6 } if (it != is.begin() && (--it)->second >= L) { L = min(L, it->first); R = max(R, it->second); is.erase(it); //05d } return is.insert(before, {L, R}); } void removeInterval(set<pii>& is, int L, int R) { //858 if (L == R) return; auto it = addInterval(is, L, R); auto r2 = it->second; if (it->first == L) is.erase(it); else (int&)it->second = L; //61f if (R != r2) is.emplace(R, r2); }</pre>	753a4c, 19 lines
<h3>ConstantIntervals.h</h3> <p>Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.</p> <p>Usage: constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});</p> <p>Time: $\mathcal{O}(k \log \frac{n}{k})$</p>	
<pre>template<class F, class G, class T> //570 void rec(int from, int to, F& f, G& g, int& i, T& p, T q) { if (p == q) return; if (from == to) { g(i, to, p); i = to; p = q; //05f } else { int mid = (from + to) >> 1; rec(from, mid, f, g, i, p, f(mid)); rec(mid+1, to, f, g, i, p, q); } //729 } template<class F, class G> void constantIntervals(int from, int to, F f, G g) { if (to <= from) return; int i = from; auto p = f(i), q = f(to-1); //a6c rec(from, to-1, f, g, i, p, q); g(i, to, q); }</pre>	
<h3>10.2 Misc. algorithms</h3> <p>TernarySearch.h</p> <p>Description: Find the smallest i in [a, b] that maximizes f(i), assuming that f(a) < ... < f(i) ≥ ... ≥ f(b). To reverse which of the sides allows non-strict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B).</p> <p>Usage: int ind = ternSearch(0, n-1, [&](int i){return a[i];});</p>	

Time: $\mathcal{O}(\log(b - a))$	9155b4, 11 lines
<pre>template<class F> //7d4 int ternSearch(int a, int b, F f) { assert(a <= b); while (b - a >= 5) { int mid = (a + b) / 2; if (f(mid) < f(mid+1)) a = mid; // (A) //ec4 else b = mid+1; } rep(i, a+1, b+1) if (f(a) < f(i)) a = i; // (B) return a; } //cbb</pre>	
<h2>10.3 Dynamic programming</h2> <p>KnuthDP.h</p> <p>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.</p> <p>Time: $\mathcal{O}(N^2)$</p>	d41d8c, 1 lines
<pre>//d41</pre>	
<h2>DivideAndConquerDP.h</h2> <p>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 $i = L..R - 1$.</p> <p>Time: $\mathcal{O}((N + (hi - lo)) \log N)$</p>	d38d2b, 18 lines
<pre>struct DP { // Modify at will: //ff9 int lo(int ind) { return 0; } int hi(int ind) { return ind; } ll f(int ind, int k) { return dp[ind][k]; } void store(int ind, int k, ll v) { res[ind] = pii(k, v); } } //ec8 void rec(int L, int R, int LO, int HI) { if (L >= R) return; int mid = (L + R) >> 1; pair<ll, int> best(LLONG_MAX, LO); rep(k, max(LO, lo(mid)), min(HI, hi(mid))) //680 best = min(best, make_pair(f(mid, k), k)); store(mid, best.second, best.first); rec(L, mid, LO, best.second+1); rec(mid+1, R, best.second, HI); } //a30 void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); } };</pre>	

<h2>10.4 Optimization tricks</h2> <p>__builtin_ia32_ldmxcsr(40896); disables denormals (which make floats 20x slower near their minimum value).</p> <h3>10.4.1 Bit hacks</h3> <ul style="list-style-type: none">x & -x is the least bit in x.for (int x = m; x;) { --x &= m; ... } loops over all subset masks of m (except m itself).c = x&-x, r = x+c; ((r^x) >> 2)/c r is the next number after x with the same number of bits set.	
---	--

- `rep(b,0,K) rep(i,0,(1 << K))`
 `if (i & 1 << b) D[i] += D[i^(1 << b)];`
 computes all sums of subsets.

10.4.2 Pragmas

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

Extra Stuff (11)

Pruefer.h
Description: Helps construct random tree Choose random n-2 length array, values [0, n-1]
Time: $\mathcal{O}(n)$

```
295d68, 23 lines
vector<pii> pruefer_decode(const vi &code) { //865
    int n = sz(code) + 2;
    vi degree(n, 1);
    for (int i : code)
        degree[i]++;
    //c77
    set<int> leaves;
    rep(i, 0, n)
        if (degree[i] == 1)
            leaves.insert(i);
    //f6a
    vector<pii> edges;
    for (int v : code) {
        int leaf = *leaves.begin();
        leaves.erase(leaves.begin());
    //1b3
        edges.emplace_back(leaf, v);
        if (--degree[v] == 1)
            leaves.insert(v);
    }
    edges.emplace_back(*leaves.begin(), n-1); //062
    return edges;
}
```

Dylan’s Templates (12)

WaveletTree.h
Description: Range K-th Smallest
Time: $\mathcal{O}(n \log^2(n))$ I think

```
ccdc27, 39 lines
struct WaveletTree { //b57
    struct Node {
        vector<int> a, b;
        int l = -1, r = -1;
        bool leaf = 0;
    }; //188
    int bits = 30;
    vector<int> arr;
    vector<Node> tree = {{{}};
    void construct(int u, int lo, int hi) {
        if (lo == hi) { tree[u].leaf = 1; return; } //5bb
        int l = sz(tree); tree.pb({});
        int r = sz(tree); tree.pb({});
        tree[u].l = l; tree[u].r = r;
    }
};
```

```
int mid = (lo + hi) / 2;
int pre = 0; //16b
for (int i = 0; i < sz(tree[u].a); i++) {
    if (arr[tree[u].a[i]] <= mid) {
        tree[l].a.pb(tree[u].a[i]);
        pre++;
    } else tree[r].a.pb(tree[u].a[i]); //b04
    tree[u].b.pb(pre);
}
if (!tree[l].a.empty()) construct(l, lo, mid);
if (!tree[r].a.empty()) construct(r, mid + 1, hi);
} //102
WaveletTree(vector<int> a) : arr(a) {
    for (int i = 0; i < sz(arr); i++) tree[0].a.pb(i);
    construct(0, 0, (1 << bits) - 1);
}
int query(int l, int r, int k, int u = 0) { //07f
    if (u == -1) return 0;
    int lp = lb(tree[u].a, l), rp = lb(tree[u].a, r + 1) - 1;
    if (tree[u].leaf) return tree[u].a[lp + k - 1];
    int n = tree[u].b[rp] - (lp == 0 ? 0 : tree[u].b[lp - 1]);
    if (n >= k) return query(l, r, k, tree[u].l); //3a4
    return query(l, r, k - n, tree[u].r);
}
};
```

12.1 Convolutions

XORConvolution.h
Description: Bitwise XOR Convolution
Time: $\mathcal{O}(n \log(n))$

```
c3f3d6, 19 lines
void xorfft(vector<ll> &v, bool inv = 0) { //475
    for (int k = 0; k < bits; k++) {
        for (int i = 0; i < 1 << bits; i++, i += i & 1 << k) {
            ll a = v[i], b = v[i ^ 1 << k];
            v[i] = a + b, v[i ^ 1 << k] = a - b;
        } //aa7
    }
    for (int i = 0; i < 1 << bits; i++) v[i] = (v[i] % mod + mod) % mod;
    if (inv) {
        ll n = 1; for (int i = 0; i < bits; i++) n = n * 2 % mod;
        n = mInv(n); //e2a
        for (int i = 0; i < 1 << bits; i++) v[i] = v[i] * n % mod;
    }
}
vector<ll> mult(vector<ll> a, vector<ll> b) {
    xorfft(a); xorfft(b); vector<ll> c(1 << bits); //703
    for (int i = 0; i < 1 << bits; i++) c[i] = a[i] * b[i] % mod;
    xorfft(c, 1); return c;
}
```