Contest (1)

template.cpp

14 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()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

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

Mathematics (2)

2.1 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k + c_1 x^{k-1} + \cdots + c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.2 Trigonometry

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

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

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

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

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

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

2.3 Geometry

2.3.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \dfrac{a+b+c}{2}$

Area: $A = \sqrt{p(p-a)(p-b)(p-c)}$

Circumradius: $R = \dfrac{abc}{4A}$

Inradius: $r = \dfrac{A}{p}$

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: $\dfrac{\sin\alpha}{a} = \dfrac{\sin\beta}{b} = \dfrac{\sin\gamma}{c} = \dfrac{1}{2R}$

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

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

2.4 Derivatives/Integrals

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

Integration by parts:

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

2.5 Sums

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

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

2.6 Series

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

Data structures (3)

OrderStatisticTree.h

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

782797, 16 lines

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

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

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

HashMap.h

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

d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
```

```
struct chash { // large odd number for C
  const uint64_t C = ll(4e18 * acos(0)) | 71;
  ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

SegmentTree.h

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

0f4bdb, 19 lines

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

LazySegmentTree.h

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

"../various/BumpAllocator.h" 34ecf5, 50 lines

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

UnionFindRollback.h

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

de4ad0, 21 lines

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

ConvexHullTrickUndo.h

Description: Convex hull trick for maximum value with undo e029e7, 46 lines

```
const bool FIRST = true, LAST = false;
template <const bool ISFIRST, class T, class F>
T bsearch(T lo, T hi, F f) {
  hi--; while (lo <= hi) {
    T mid = lo+(hi-lo)/2; if (f(mid) == ISFIRST) hi = mid-1;
    else lo = mid+1;
  }
  return ISFIRST ? lo : hi;
}

template <class T, class Cmp = less<T>>
struct ConvexHullTrickUndo {
  struct Line {
    T m, b; Line(T m = T(), T b = T()) : m(m), b(b) {}
    T eval(T x) const { return m * x + b; }
  };
  vector<pair<int, Line>> history; vector<Line> L;
  Cmp cmp; int back;
  ConvexHullTrickUndo(Cmp cmp = Cmp()) : cmp(cmp), back(0) {}
  int size() const { return back; }
  void addLine(T m, T b) {
    int i = back; if (i >= 1)
      i = bsearch<LAST>(1, i + 1, [&] (int j) {
        return cmp(m, L[j - 1].m) || cmp(L[j - 1].m, m)
            || cmp(b, L[j - 1].b);
      });
    if (i >= 2)
      i = bsearch<LAST>(2, i + 1, [&] (int j) {
        T c1 = (L[j - 1].m - m) * (L[j - 2].b - b);
        T c2 = (L[j - 1].b - b) * (L[j - 2].m - m);
        return c1 > c2;
      });
    if (i == int(L.size())) L.emplace_back();
    history.emplace_back(back, L[i]);
    L[i] = Line(m, b); back = i + 1;
  }
  void undo() {
    tie(back,L[back - 1]) = history.back(); history.pop_back();
  }
  T getMax(T x) const {
    return L[bsearch<FIRST>(0, back - 1, [&] (int i) {
      return cmp(L[i + 1].eval(x), L[i].eval(x));
    })].eval(x);
  }
  void reserve(int N) { L.reserve(N); history.reserve(N); }
};
```

LineContainer.h

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

8ec1c7, 30 lines

```
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

LiChaoTree.h

Description: Supports adding lines or line segments

5982b2, 52 lines

```
template <class T, class Cmp = less<T>> struct LiChaoTree {
  using Line = pair<T, T>; int N; Cmp cmp; T INF;
  vector<Line> TR; vector<T> X;
  T eval(Line l, int i) const { return l.first*X[i]+l.second; }
  bool majorize(Line a, Line b, int l, int r) {
    return !cmp(eval(a, l), eval(b, l))
        && !cmp(eval(a, r), eval(b, r));
  }
  int cInd(T x) const {
    return lower_bound(X.begin(), X.end(), x) - X.begin();
  }
  int fInd(T x) const {
    return upper_bound(X.begin(), X.end(), x) - X.begin() - 1;
  }
  LiChaoTree(const vector<T> &xs, Cmp cmp = Cmp(),
             T inf = numeric_limits<T>::max())
      : cmp(cmp), INF(min(inf, -inf, cmp)), X(xs) {
    sort(X.begin(), X.end());
    X.erase(unique(X.begin(), X.end()), X.end());
    N = X.size();
    TR.assign(N == 0 ? 0 : 1 << __lg(N*4-1), Line(T(), INF));
  }
  void addLine(int k, int tl, int tr, Line line) {
    if (majorize(line, TR[k], tl, tr)) swap(line, TR[k]);
    if (majorize(TR[k], line, tl, tr)) return;
    if (cmp(eval(TR[k], tl), eval(line, tl))) swap(line,TR[k]);
    int m=tl+(tr-tl)/2; if (!cmp(eval(line,m),eval(TR[k],m))) {
      swap(line, TR[k]); addLine(k * 2, tl, m, line);
    } else addLine(k * 2 + 1, m + 1, tr, line);
  }
  void addLineSegment(
      int k, int tl, int tr, int l, int r, Line line) {
    if (r < tl || tr < l) return;
    if (l <= tl && tr <= r) {addLine(k, tl, tr, line); return;}
    int m = tl + (tr - tl) / 2;
    addLineSegment(k * 2, tl, m, l, r, line);
    addLineSegment(k * 2 + 1, m + 1, tr, l, r, line);
  }
  T getMax(int k, int tl, int tr, int i) const {
    T ret = eval(TR[k], i); if (tl == tr) return ret;
    int m = tl + (tr - tl) / 2;
    if (i <= m) return max(ret, getMax(k * 2, tl, m, i), cmp);
    else return max(ret, getMax(k * 2 + 1, m + 1, tr, i), cmp);
  }
  void addLine(T m, T b) { addLine(1, 0, N - 1, Line(m, b)); }
  void addLineSegment(T m, T b, T l, T r) {
    int li = cInd(l), ri = fInd(r); if (li <= ri)
      addLineSegment(1, 0, N - 1, li, ri, Line(m, b));
  }
  T getMax(T x) const { return getMax(1, 0, N - 1, cInd(x)); }
  void clear() { fill(TR.begin(), TR.end(), Line(T(), INF)); }
};
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.

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

e62fac, 22 lines

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

FenwickTree2d.h

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

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

"FenwickTree.h" 157f07, 22 lines

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

RMQ.h

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

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

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

510c32, 16 lines

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

MoQueries.h

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

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

a12ef4, 49 lines

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

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

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

Splay.h

Description: Splay Tree

1e50b7, 124 lines

```
template <class C> struct NodeLazyAgg {
  using Data = typename C::Data; using Lazy = typename C::Lazy;
  bool rev; int sz; NodeLazyAgg *l, *r, *p;
  Lazy lz; Data val, sbtr;
  NodeLazyAgg(const Data &v)
      : rev(false), sz(1), l(nullptr), r(nullptr), p(nullptr),
        lz(C::ldef()), val(v), sbtr(v) {}
  void update() {
    sz = 1; sbtr = val;
    if (l) { sz += l->sz; sbtr = C::merge(l->sbtr, sbtr); }
    if (r) { sz += r->sz; sbtr = C::merge(sbtr, r->sbtr); }
  }
  void propagate() {
    if (rev) {
      if (l) l->reverse();
      if (r) r->reverse();
      rev = false;
    }
    if (lz != C::ldef()) {
      if (l) l->apply(lz);
      if (r) r->apply(lz);
      lz = C::ldef();
    }
  }
  void apply(const Lazy &v) {
    lz = C::mergeLazy(lz, v); val = C::applyLazy(val, v, 1);
    sbtr = C::applyLazy(sbtr, v, sz);
  }
  void reverse() { rev = !rev; swap(l, r); C::revData(sbtr); }
  static Data qdef() { return C::qdef(); }
};

template <class _Node, class Container = deque<_Node>>
struct Splay {
  using Node = _Node; Container TR; deque<Node *> del;
  template <class T> Node *makeNode(const T &v) {
    if (del.empty()) { TR.emplace_back(v); return &TR.back(); }
    Node *x = del.back(); del.pop_back();
    *x = typename Container::value_type(v); return x;
  }
  bool isRoot(Node *x) {
    return !x->p || (x != x->p->l && x != x->p->r);
  }
  void con(Node *x, Node *p, bool hasCh, bool isL) {
    if (x) x->p = p;
    if (hasCh) (isL ? p->l : p->r) = x;
  }
  void rot(Node *x) {
    Node *p = x->p, *g = p->p;
    bool isRootP = isRoot(p), isL = x == p->l;
    con(isL ? x->r : x->l, p, true, isL);
    con(p, x, true, !isL);
    con(x, g, !isRootP, !isRootP && p == g->l); p->update();
  }
  void splay(Node *x) {
    while (!isRoot(x)) {
      Node *p = x->p, *g = p->p; g->propagate();
      p->propagate(); x->propagate();
      if (!isRoot(p)) rot((x == p->l) == (p == g->l) ? p : x);
      rot(x);
    }
    x->propagate(); x->update();
  }
  template <class F>
  void applyToRange(Node *&root, int i, int j, F f) {
    if (i == 0 && j == (root ? root->sz : 0) - 1) {
      f(root); if (root) { root->propagate(); root->update(); }
    } else {
      Node *t = i ? select(root, i - 1)->r
                  : select(root, j + 1)->l;
      con(nullptr, root, true, !i); root->update();
      con(t, nullptr, false, !i); applyToRange(t, 0, j - i, f);
      con(t, root, true, !i); root->update();
    }
  }
  Node *select(Node *&root, int k) {
    Node *last = nullptr; while (root) {
      (last = root)->propagate();
      int t = root->l ? root->l->sz : 0;
      if (t > k) root = root->l;
      else if (t < k) { root = root->r; k -= t + 1; }
      else break;
    }
    if (last) splay(root = last);
    return root;
  }
  int index(Node *&root, Node *x) {
    root = x; if (!root) return -1;
    splay(root); return root->l ? root->l->sz : 0;
  }
  template <class T, class Comp>
  pair<int, Node *> getFirst(Node *&root, const T &v,
                             Comp cmp) {
    pair<int, Node *> ret(0, nullptr); Node *last = nullptr;
    while (root) {
      (last = root)->propagate();
      if (!cmp(root->val, v)) {
        ret.second = root; root = root->l;
      } else {
        ret.first += 1 + (root->l ? root->l->sz : 0);
        root = root->r;
      }
    }
    if (last) splay(root = last);
    return ret;
  }
  template <class T>
  Node *buildRec(const vector<T> &A, int l, int r) {
    if (l > r) return nullptr;
    int m = l + (r - l) / 2;
    Node *left = buildRec(A, l, m - 1);
    Node *ret = makeNode(A[m]), *right = buildRec(A, m + 1, r);
    con(left, ret, ret, true); con(right, ret, ret, false);
    ret->update(); return ret;
  }
  template <class T>
  Node *build(const vector<T> &A) {
    return buildRec(A, 0, int(A.size()) - 1);
  }
  void clear(Node *x) {
    if (!x) return;
    clear(x->l); del.push_back(x); clear(x->r);
  }
};
```

LCT.h

Description: Link Cut Tree

"Splay.h" f48b54, 87 lines

```
template <class Node>
struct LCT : public Splay<Node, vector<Node>> {
  using Tree = Splay<Node, vector<Node>>; using Tree::TR;
  using Tree::makeNode; using Tree::splay; using Tree::select;
  using Data = typename Node::Data;
  using Lazy = typename Node::Lazy;
  int vert(Node *x) { return x - TR.data(); }
  Node *access(Node *x) {
    Node *last = nullptr; for (Node *y = x; y; y = y->p) {
      splay(y); y->r = last; last = y;
    }
    splay(x); return last;
  }
  void makeRoot(Node *x) { access(x); x->reverse(); }
  Node *findMin(Node *x) {
    for (x->propagate(); x->l; (x = x->l)->propagate());
    splay(x); return x;
  }
  Node *findMax(Node *x) {
    for (x->propagate(); x->r; (x = x->r)->propagate());
    splay(x); return x;
  }
  void makeRoot(int x) { makeRoot(&TR[x]); }
  int lca(int x, int y) {
    if (x == y) return x;
    access(&TR[x]); Node *ny = access(&TR[y]);
    return TR[x].p ? vert(ny) : -1;
  }
  bool connected(int x, int y) { return lca(x, y) != -1; }
  void link(int x, int y) { makeRoot(y); TR[y].p = &TR[x]; }
  bool safeLink(int x, int y) {
    if (connected(x, y)) return false;
    link(x, y); return true;
  }
  bool linkParent(int par, int ch) {
    access(&TR[ch]); if (TR[ch].l) return false;
    TR[ch].p = &TR[par]; return true;
  }
  bool cut(int x, int y) {
    makeRoot(x); access(&TR[y]);
    if (&TR[x] != TR[y].l || TR[x].r) return false;
    TR[y].l->p = nullptr; TR[y].l = nullptr; return true;
  }
  bool cutParent(int x) {
    access(&TR[x]); if (!TR[x].l) return false;
    TR[x].l->p = nullptr; TR[x].l = nullptr; return true;
  }
  int findParent(int x) {
    access(&TR[x]);
    return TR[x].l ? vert(findMax(TR[x].l)) : -1;
  }
  int findRoot(int x) {
    access(&TR[x]); return vert(findMin(&TR[x]));
  }
  int depth(int x) {
    access(&TR[x]); return TR[x].l ? TR[x].l->sz : 0;
  }
  int kthParent(int x, int k) {
    int d = depth(x); Node *nx = &TR[x];
    return k <= d ? vert(select(nx, d - k)) : -1;
  }
  void updateVertex(int x, const Lazy &v) {
    access(&TR[x]); Node *l = TR[x].l; TR[x].l = nullptr;
    TR[x].apply(v); TR[x].propagate(); TR[x].update();
    TR[x].l = l;
  }
  void updatePathFromRoot(int to, const Lazy &v) {
    access(&TR[to]); TR[to].apply(v);
  }
  void updatePath(int from, int to, const Lazy &v) {
    makeRoot(from); access(&TR[to]);
    if (from != to && !TR[from].p) return false;
    TR[to].apply(v); return true;
  }
  Data queryVertex(int x) { access(&TR[x]); return TR[x].val; }
  Data queryPathFromRoot(int to) {
    access(&TR[to]); return TR[to].sbtr;
  }
  Data queryPath(int from, int to) {
    makeRoot(from); access(&TR[to]);
    return from == to || TR[from].p
        ? TR[to].sbtr : Node::qdef();
  }
  LCT(const vector<Data> &A) {
    TR.reserve(A.size()); for (auto &&a : A) makeNode(a);
  }
};
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

c9b7b0, 17 lines

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

PolyRoots.h

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

"Polynomial.h" b00bfe, 23 lines

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

PolyInterpolate.h

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

08bf48, 13 lines

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

BerlekampMassey.h

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

"../number-theory/ModPow.h" 96548b, 20 lines

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

  ll b = 1;
  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;
    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;
  }

  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 $S[0 \ldots \geq n-1]$ and $tr[0 \ldots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`
Time: $\mathcal{O}\left(n^2 \log k\right)$

f4e444, 26 lines

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

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

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

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

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

4.2 Optimization

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.
Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`
Time: $\mathcal{O}\left(\log((b-a)/\epsilon)\right)$

31d45b, 14 lines

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

HillClimbing.h

Description: Poor man's optimization for unimodal functions 8eeeaf, 14 lines

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

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

IntegrateAdaptive.h

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

92dd79, 15 lines

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

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

Simplex.h

Description: Two Phase Simplex to solve a linear programming problem with N variables and M equations in canonical form max c^T x, subject to Ax <= b and x >= 0, where A is an M x N matrix; b is a vector with dimension M; c, x are vectors with dimension N. val is the value of the maximum cost of the objective function c^T x, INF if unbounded, -INF if infeasible. x is a solution vector of dimension N producing the maximum cost, can any solution if unbounded, empty if infeasible.

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

ae8f9f, 69 lines

```
template <class F> struct Simplex {
  int M, N; F INF, EPS, val; vector<int> IN, OUT; vector<F> x;
  vector<vector<F>> T;
  bool cmp(F a, int b, F c, int d) {
    return abs(a - c) > EPS ? a < c : b < d;
  }
  void pivot(int r, int s) {
    auto &a1 = T[r]; F inv1 = F(1) / a1[s];
    for (int i = 0; i <= M + 1; i++)
      if (i != r && abs(T[i][s]) > EPS) {
        auto &a2 = T[i]; F inv2 = a2[s] * inv1;
        for (int j = 0; j <= N + 1; j++) a2[j] -= a1[j] * inv2;
        a2[s] = a1[s] * inv2;
      }
    for (int j = 0; j <= N + 1; j++) if (j != s) a1[j] *= inv1;
    for (int i = 0; i <= M + 1; i++) if (i != r)
      T[i][s] *= -inv1;
    a1[s] = inv1; swap(IN[r], OUT[s]);
  }
  bool simplex(int phase) {
    int x = M + phase - 1; while (true) {
      int s = -1; for (int j = 0; j <= N; j++)
        if (OUT[j] != -phase
            && (s == -1
                || cmp(T[x][j], OUT[j], T[x][s], OUT[s])))
          s = j;
      if (T[x][s] >= -EPS) return true;
      int r = -1; for (int i = 0; i < M; i++)
        if (T[i][s] > EPS
            && (r == -1 || cmp(T[i][N + 1] * T[r][s], IN[i],
                               T[r][N + 1] * T[i][s], IN[r])))
          r = i;
      if (r == -1) return false;
      pivot(r, s);
    }
  }
  Simplex(vector<vector<F>> A, const vector<F> &b,
          const vector<F> &c, F INF, F EPS)
    : M(b.size()), N(c.size()), INF(INF), EPS(EPS), IN(M),
      OUT(N + 1), T(move(A)) {
    T.reserve(M + 2); for (int i = 0; i < M; i++) {
      T[i].resize(N + 2, F());
      IN[i] = N + i; T[i][N] = F(-1); T[i][N + 1] = b[i];
    }
    T.emplace_back(N + 2, F()); T.emplace_back(N + 2, F());
    for (int j = 0; j < N; j++) {
      OUT[j] = j; T[M][j] = -c[j];
    }
    OUT[N] = -1; T[M + 1][N] = F(1); int r = 0;
    for (int i = 1; i < M; i++) if (T[i][N + 1] < T[r][N + 1])
      r = i;
    if (T[r][N + 1] < -EPS) {
      pivot(r, N);
      if (!simplex(2) || T[M + 1][N + 1] < -EPS) {
        val = -INF; return;
      }
      for (int i = 0; i < M; i++) if (IN[i] == -1) {
        int s = 0; for (int j = 1; j <= N; j++)
          if (s == -1 || cmp(T[i][j], OUT[j], T[i][s], OUT[s]))
            s = j;
        pivot(i, s);
      }
    }
    bool unbounded = !simplex(1); x.assign(N, F());
    for (int i = 0; i < M; i++) if (IN[i] < N)
      x[IN[i]] = T[i][N + 1];
    val = unbounded ? INF : T[M][N + 1];
  }
};
```

4.3 Matrices

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.

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

bd5cec, 15 lines

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

IntDeterminant.h

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

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

3313dc, 18 lines

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

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.

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

44c9ab, 38 lines

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

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

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

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

SolveLinear2.h

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

"SolveLinear.h" 08e495, 7 lines

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

SolveLinearBinary.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.

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

fa2d7a, 34 lines

```
typedef bitset<1000> bs;

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

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

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (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 mod p, and k is doubled in each step.

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

ebfff6, 35 lines

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

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

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

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

Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

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

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/FFTMod.

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

00ced6, 35 lines

```
typedef complex<double> C;
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)
  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];
  }
  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) {
      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;
    }
}
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);
  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]);
  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;
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);
  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);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  }
  fft(outl), fft(outs);
  rep(i,0,sz(res)) {
    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;
}
```

NumberTheoreticTransform.h

Description: ntt(a) computes $\hat{f}(k) = \sum_x a[x]g^{xk}$ for all k, where $g = \text{root}^{(mod-1)/N}$. 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, 33 lines

```
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// 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);
  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;
  }
  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) {
      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);
    }
}
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;
  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) 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) {
  for (int n = sz(a), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
      int &u = a[j], &v = a[j + step]; tie(u, v) =
        inv ? pii(v - u, u) : pii(v, u + v); // AND
        inv ? pii(v, u - v) : pii(u + v, u); // OR
        pii(u + v, u - v);                   // XOR
    }
  }
  if (inv) for (int& x : a) x /= sz(a); // XOR only
}
vi conv(vi a, vi b) {
  FST(a, 0); FST(b, 0);
  rep(i,0,sz(a)) a[i] *= b[i];
  FST(a, 1); return a;
}
```

Number theory (5)

5.1 Modular arithmetic

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM $\leq$ mod and that mod is a prime. For a prime p, can also be computed as modpow(a, p-2).

6f684f, 3 lines

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

ModPow.h

b83e45, 8 lines

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

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

ModLog.h

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

c040b8, 11 lines

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

ModSum.h

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

5c5bc5, 16 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

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

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);
}
```

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) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1); // else no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

5.2 Primality

FastEratosthenes.h

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

6b2912, 20 lines

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

MillerRabin.h

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

"ModMulLL.h" 60dcd1, 12 lines

```
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
      s = __builtin_ctzll(n-1), d = n >> s;
  for (ull a : A) {   // ^ count trailing zeroes
    ull p = modpow(a%n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--)
      p = modmul(p, p, n);
    if (p != n-1 && i != s) return 0;
  }
  return 1;
}
```

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.

"ModMulLL.h", "MillerRabin.h" a33cf6, 18 lines

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

```
vector<ull> factor(ull n) {
  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));
  return l;
}
```

5.3 Divisibility

euclid.h

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

33ba8f, 5 lines

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

CRT.h

Description: Chinese Remainder Theorem.

crt(a, m, b, n) computes x such that $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$. If $|a| < m$ and $|b| < n$, x will obey $0 \le x < \mathrm{lcm}(m,n)$. Assumes $mn < 2^{62}$.

Time: $\log(n)$

"euclid.h" 04d93a, 7 lines

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y, g = euclid(m, n, x, y);
  assert((a - b) % g == 0); // else no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m*n/g : x;
}
```

5.3.1 Bézout's identity

For $a \neq, b \neq 0$, 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 $\le n$ 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 $\phi(n) = (p_1-1)p_1^{k_1-1}...(p_r-1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1-1/p)$.

$\sum_{d|n}\phi(d) = n$, $\sum_{1 \le k \le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$

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

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

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

5.4 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \ge 0$, finds the closest rational approximation p/q with $p,q \le N$. It will obey $|p/q - x| \le 1/qN$.

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

dd6c5e, 21 lines

```
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
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),
       NP = b*P + LP, NQ = b*Q + LQ;
    if (a > b) {
      // If b > a/2, we have a semi-convergent that gives us a
      // better approximation; if b = a/2, we *may* have one.
      // Return {P, Q} here for a more canonical approximation.
      return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
        make_pair(NP, NQ) : make_pair(P, Q);
    }
    if (abs(y = 1/(y - (d)a)) > 3*N) {
      return {NP, NQ};
    }
    LP = P; P = NP;
    LQ = Q; Q = NQ;
  }
}
```

FracBinarySearch.h

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

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

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

27ab3e, 25 lines

```
struct Frac { ll p, q; };

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

5.5 Estimates

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

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

5.6 Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

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

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

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

5.7 Other

SumFloorLinear.h

Description: Computes the sum of floor((a * i + b) / m) for 0 <= i < n Equivalent to the number of integer points (x, y) where 0 <= x < n and 0 <= y <= (a * i + b) / m

f1c0a0, 7 lines

```
template <class T> T sumLinear(T n, T a, T b, T m) {
  if (a == 0) return (b / m) * n;
  if (a >= m || b >= m)
    return ((a / m) * (n - 1) + 2 * (b / m)) * n / 2
        + sumLinear(n, a % m, b % m, m);
  return sumLinear((a * n + b) / m, m, (a * n + b) % m, a);
}
```

MatroidIntersection.h

Description: Find the largest subset of a ground set of N elements that is independent in both matroids

4704d9, 46 lines

```
template <class Matroid1, class Matroid2>
struct MatroidIntersection {
  int N; vector<bool> inSet; vector<int> independentSet;
  bool augment(Matroid1 &m1, Matroid2 &m2) {
    vector<int> par(N + 1, -1); queue<int> q; q.push(N);
    while (!q.empty()) {
      int v = q.front(); q.pop(); if (inSet[v]) {
        m1.clear(); for (int i = 0; i < N; i++)
          if (inSet[i] && i != v) m1.add(i);
        rep(i, 0, N)
          if (!inSet[i] && par[i] == -1 && m1.independent(i)) {
            par[i] = v; q.push(i);
          }
      } else {
        auto backE = [&] {
          m2.clear(); rep(c, 0, 2) rep(i, 0, N)
            if ((v == i || inSet[i]) && (par[i] == -1) == c) {
              if (!m2.independent(i)) {
                if (c) { par[i] = v; q.push(i); return i; }
                else return -1;
              }
              m2.add(i);
            }
          return N;
        };
        for (int w = backE(); w != -1; w = backE())
```

```
          if (w == N) {
            for (; v != N; v = par[v]) inSet[v] = !inSet[v];
            return true;
          }
      }
    }
    return false;
  }
  MatroidIntersection(int N, Matroid1 m1, Matroid2 m2)
      : N(N), inSet(N + 1, false) {
    m1.clear(); m2.clear(); inSet[N] = true;
    for (int i = N - 1; i >= 0; i--)
      if (m1.independent(i) && m2.independent(i)) {
        inSet[i] = true; m1.add(i); m2.add(i);
      }
    while (augment(m1, m2));
    inSet.pop_back();
    rep(i, 0, N) if (inSet[i]) independentSet.push_back(i);
  }
};
```

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

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

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 Burnside's lemma

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

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

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

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

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

6.2.2 Lucas' Theorem

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

6.2.3 Binomials

multinomial.h

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

a0a312, 6 lines

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

6.3 General purpose numbers

6.3.1 Bernoulli numbers

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty}\frac{B_k}{k!}f^{(k-1)}(m)$$

$$\approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

6.3.3 Eulerian numbers

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

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 Bell numbers

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

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

6.3.6 Labeled unrooted trees

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

6.3.7 Catalan numbers

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

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

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

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).

- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Graph (7)

7.1 Fundamentals

BellmanFord.h

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

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

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

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

FloydWarshall.h

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

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

7.2 Network flow

PushRelabel.h

Description: Push-relabel flow
Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$ 0aaa51, 51 lines

```
template <class _FU> struct FlowEdge {
  using FU = _FU; int to, rev; FU cap, resCap;
};
template <class Ed> struct PushRelabelMaxFlow {
  using FU = typename Ed::FU;
  int V; FU EPS; vector<vector<Ed>> G; vector<bool> cut;
  PushRelabelMaxFlow(int V, FU EPS)
      : V(V), EPS(EPS), G(V) {}
  void addEdge(int v, int w, FU vwCap, FU wvCap) {
    if (v == w) return;
    G[v].emplace_back(w, int(G[w].size()), vwCap);
    G[w].emplace_back(v, int(G[v].size()) - 1, wvCap);
  }
  FU getFlow(int s, int t) {
    cut.assign(V, false); if (s == t) return FU();
    vector<FU> ex(V, FU()); vector<int> h(V, 0), cnt(V * 2, 0);
    vector<vector<int>> hs(V * 2);
    vector<typename vector<Ed>::iterator> cur(V);
    auto push = [&] (int v, Ed &e, FU df) {
      int w = e.to; if (abs(ex[w]) <= EPS && df > EPS)
        hs[h[w]].push_back(w);
      e.resCap -= df; G[w][e.rev].resCap += df;
      ex[v] -= df; ex[w] += df;
    };
    h[s] = V; ex[t] = FU(1); cnt[0] = V - 1;
    rep(v, 0, V) cur[v] = G[v].begin();
    for (auto &&e : G[s]) push(s, e, e.resCap);
    if (!hs[0].empty()) for (int hi = 0; hi >= 0;) {
      int v = hs[hi].back(); hs[hi].pop_back();
      while (ex[v] > EPS) {
        if (cur[v] == G[v].end()) {
          h[v] = INT_MAX;
          for (auto e = G[v].begin(); e != G[v].end(); e++)
            if (e->resCap > EPS && h[v] > h[e->to] + 1)
              h[v] = h[(cur[v] = e)->to] + 1;
          cnt[h[v]]++; if (--cnt[hi] == 0 && hi < V)
            rep(w, 0, V) if (hi < h[w] && h[w] < V) {
              cnt[h[w]]--; h[w] = V + 1;
            }
          hi = h[v];
        } else if (cur[v]->resCap > EPS
            && h[v] == h[cur[v]->to] + 1) {
          push(v, *cur[v], min(ex[v], cur[v]->resCap));
        } else cur[v]++;
      }
      while (hi >= 0 && hs[hi].empty()) hi--;
    }
    rep(v, 0, V) cut[v] = h[v] >= V;
    return -ex[s];
  }
};
```

MinCostMaxFlow.h

Description: Min-cost max-flow. cap[i][j] != cap[j][i] is allowed; double edges are not. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.
Time: Approximately $\mathcal{O}(E^2)$ fe85cc, 81 lines

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

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

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

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

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

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

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

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

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

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

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

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph
Time: $\mathcal{O}(VE\log V)$

e0d723, 59 lines

```
template <class T> struct StoerWagnerGlobalMinCut {
  struct Edge {
    int to, rev; T wt;
    Edge(int to, int rev, T wt) : to(to), rev(rev), wt(wt) {}
  };
  struct Node {
    T w; int v; Node(T w, int v) : w(w), v(v) {}
    bool operator < (const Node &o) const { return w < o.w; }
  };
  int V; vector<vector<Edge>> G; vector<bool> cut;
  T cutWeight, INF;
  void addEdge(int v, int w, T wt) {
    if (v == w) return;
    G[v].emplace_back(w, int(G[w].size()), wt);
    G[w].emplace_back(v, int(G[v].size()) - 1, wt);
  }
  StoerWagnerGlobalMinCut(int V, T INF)
      : V(V), G(V), cut(V, false), cutWeight(INF), INF(INF) {}
  T globalMinCut() {
    vector<vector<Edge>> H = G;
    fill(cut.begin(), cut.end(), false);
    cutWeight = INF; vector<int> par(V);
    iota(par.begin(), par.end(), 0);
    for (int phase = V - 1; phase > 0; phase--) {
      vector<T> W(V, T()); __gnu_pbds::priority_queue<Node> PQ;
      vector<typename decltype(PQ)::point_iterator> ptr(
          V, PQ.end());
      rep(v, 1, V) if (par[v] == v)
        ptr[v] = PQ.push(Node(W[v], v));
      for (auto &&e : H[0]) if (ptr[e.to] != PQ.end())
        PQ.modify(ptr[e.to], Node(W[e.to] += e.wt, e.to));
      for (int i = 0, v, last = 0; i < phase; i++, last = v) {
        T w = PQ.top().w; v = PQ.top().v; PQ.pop();
        ptr[v] = PQ.end(); if (i == phase - 1) {
          if (cutWeight > w) {
            cutWeight = w; rep(x, 0, V) cut[x] = par[x] == v;
          }
          fill(W.begin(), W.end(), T());
          for (auto &&e : H[v]) W[e.to] += e.wt;
          for (auto &&e : H[last]) {
            e.wt += W[e.to]; H[e.to][e.rev].wt += W[e.to];
            W[e.to] = T();
          }
          for (auto &&e : H[v]) if (W[e.to] != T()) {
            H[e.to][e.rev].to = last;
            H[e.to][e.rev].rev = H[last].size();
            H[last].emplace_back(e.to, e.rev, e.wt);
          }
          H[v].clear();
          rep(x, 0, V) if (par[x] == v) par[x] = last;
        } else {
          for (auto &&e : H[v]) if (ptr[e.to] != PQ.end())
            PQ.modify(ptr[e.to], Node(W[e.to] += e.wt, e.to));
        }
      }
    }
    return cutWeight;
  }
};
```

GomoryHu.h

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

"HopcroftKarp.h" 1cbec6, 14 lines

```
template <class T> struct GomoryHu {
  using Edge = tuple<int, int, T>; vector<Edge> treeEdges;
  GomoryHu(int V, const vector<Edge> &edges, T EPS) {
    PushRelabelMaxFlow<FlowEdge<T>> mf(V, EPS);
    for (auto &&e : edges)
      mf.addEdge(get<0>(e), get<1>(e), get<2>(e), get<2>(e));
    vector<int> par(V, 0); rep(i, 1, V) {
      rep(v, 0, V) for (auto &&e : mf.G[v]) e.resCap = e.cap;
      treeEdges.emplace_back(i, par[i], mf.getFlow(i, par[i]));
      rep(j, i + 1, V) if (par[j] == par[i] && mf.cut[j])
        par[j] = i;
    }
  }
};
```

7.3 Matching

HopcroftKarp.h

Description: Computes the maximum matching (and minimum vertex cover) on an unweighted bipartite graph. The maximum independent set is any vertex not in the minimum vertex cover. mate is the other vertex in the matching, or -1 if unmatched. inCover is whether a vertex is in the minimum vertex cover or not.
Time: $\mathcal{O}\left(\sqrt{V}E\right)$

89608c, 41 lines

```
struct HopcroftKarpMaxMatch {
  int V, cardinality; vector<int> mate, lvl, q, type0;
  vector<bool> co, inCover, vis;
  template <class BG> bool bfs(const BG &G) {
    int front = 0, back = 0; for (int v : type0) {
      if (mate[v] == -1) lvl[q[back++] = v] = 0;
      else lvl[v] = -1;
    }
    while (front < back) {
      int v = q[front++]; for (int w : G[v]) {
        if (mate[w] == -1) return true;
        else if (lvl[mate[w]] == -1)
          lvl[q[back++] = mate[w]] = lvl[v] + 1;
      }
    }
    return false;
  }
  template <class BG> bool dfs(const BG &G, int v) {
    for (int w : G[v])
      if (mate[w] == -1
          || (lvl[mate[w]] == lvl[v] + 1 && dfs(G, mate[w]))) {
        mate[mate[v] = w] = v; return true;
      }
    lvl[v] = -1; return false;
  }
  template <class BG> void dfs2(const BG &G, int v) {
    if (vis[v]) return;
    vis[v] = true;
    for (int w : G[v]) if ((mate[v] == w) == co[v]) dfs2(G, w);
  }
  template <class BG>
  HopcroftKarpMaxMatch(const BG &G, const vector<bool> &co)
      : V(G.size()), cardinality(0), mate(V, -1), lvl(V), q(V),
        co(co), inCover(V, false), vis(V, false) {
    rep(v, 0, V) if (!co[v]) type0.push_back(v);
    while (bfs(G)) for (int v : type0)
      if (mate[v] == -1 && dfs(G, v)) cardinality++;
    for (int v : type0) if (mate[v] == -1) dfs2(G, v);
    for (int v = 0; v < V; v++) inCover[v] = vis[v] == co[v];
  }
};
```

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: $\mathcal{O}\left(N^2M\right)$

1e0fe9, 31 lines

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

GeneralMatching.h

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

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

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

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

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

StableMarriage.h

Description: Given N people of type A and M people of type B, and a list of their ranked preferences for partners, the goal is to arrange min(N, M) pairs such that if a person x of type A prefers a person y of type B more than their current partner, then person y prefers their current partner more than x.

716bdd, 24 lines

```
struct StableMarriage {
  int N, M; vector<int> bForA, aForB;
  StableMarriage(const vector<vector<int>> &aPrefs,
                 const vector<vector<int>> &bPrefs)
    : N(aPrefs.size()), M(bPrefs.size()),
      bForA(N, -1), aForB(M, -1) {
    bool rv = N > M; if (rv) { swap(N, M); bForA.swap(aForB); }
    auto &A = rv ? bPrefs : aPrefs, &B = rv ? aPrefs : bPrefs;
    vector<vector<int>> bRankOfA(M, vector<int>(N));
    rep(b, 0, M) rep(a, 0, N) bRankOfA[b][B[b][a]] = a;
    queue<int> q; rep(a, 0, N) q.push(a);
    vector<int> cur(N, 0); while (!q.empty()) {
      int a = q.front(); q.pop(); while (true) {
        int b = A[a][cur[a]++];
        if (aForB[b] == -1) { aForB[b] = a; break; }
        else if (bRankOfA[b][a] < bRankOfA[b][aForB[b]]) {
          q.push(aForB[b]); aForB[b] = a; break;
        }
      }
    }
    rep(b, 0, M) if (aForB[b] != -1) bForA[aForB[b]] = b;
    if (rv) { swap(N, M); bForA.swap(aForB); }
  }
};
```

StableRoommates.h

Description: Given N people, and a list of their ranked preferences for roommates, the goal is to arrange N / 2 pairs such that if a person x prefers a person y more than their current roommate, then person y prefers their current roommate more than x.

7046a1, 65 lines

```
struct StableRoommates {
  struct NoMatch {}; int N; vector<int> mate;
  StableRoommates(vector<vector<int>> prefs)
    : N(prefs.size()), mate(N, -1) {
    if (N % 2 == 1 || N <= 0) return;
    vector<vector<int>> rnk(N, vector<int>(N, 0));
    vector<int> fr(N, 0), bk(N, N - 1), proposed(N, -1);
    vector<vector<bool>> active(N, vector<bool>(N, true));
    queue<int> q;
    auto rem = [&] (int i, int j) {
      active[i][j] = active[j][i] = false;
    };
    auto clip = [&] (int i) {
      while (fr[i] < bk[i] && !active[i][prefs[i][fr[i]]])
        fr[i]++;
      while (fr[i] < bk[i] && !active[i][prefs[i][bk[i] - 1]])
        bk[i]--;
      if (fr[i] >= bk[i]) throw NoMatch();
    };
    auto add = [&] (int i, int j) {
      proposed[mate[i] = j] = i; while (true) {
        clip(j); if (prefs[j][bk[j] - 1] != i)
          rem(j, prefs[j][bk[j] - 1]);
        else break;
      }
    };
    auto nxt = [&] (int i) {
      clip(i); int j = prefs[i][fr[i]++];
      clip(i); prefs[i][--fr[i]] = j;
      return proposed[prefs[i][fr[i] + 1]];
    };
    rep(i, 0, N) {
      q.push(i); rep(j, 0, N - 1) rnk[i][prefs[i][j]] = j;
    }
    try {
      while (!q.empty()) {
        int i = q.front(); q.pop(); while (true) {
          clip(i); int j = prefs[i][fr[i]], i2 = proposed[j];
          if (i2 != -1 && rnk[j][i2] < rnk[j][i]) {
            rem(i, j); continue;
          }
          if (i2 != -1) {
            mate[i2] = proposed[j] = -1; q.push(i2);
          }
          add(i, j); break;
        }
      }
      int cur = 0; while (true) {
        for (; cur < N; cur++) {
          clip(cur); if (bk[cur] - fr[cur] > 1) break;
        }
        if (cur == N) break;
        vector<int> cyc1, cyc2; int i = cur, j = i;
        do { i = nxt(i); j = nxt(nxt(j)); } while (i != j);
        do { cyc1.push_back(j); j = nxt(j); } while (i != j);
        for (int k : cyc1) {
          j = mate[k]; cyc2.push_back(j);
          mate[k] = proposed[j] = -1; rem(k, j);
        }
        for (int k = 0; k < int(cyc1.size()); k++)
          add(cyc1[k], cyc2[(k + 1) % cyc2.size()]);
      }
    } catch (NoMatch &) { fill(mate.begin(), mate.end(), -1); }
  }
};
```

7.4 DFS algorithms

SCC.h

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

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

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

76b5c9, 24 lines

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

  if (low == val[j]) {
    do {
      x = z.back(); z.pop_back();
      comp[x] = ncomps;
      cont.push_back(x);
    } while (x != j);
    f(cont); cont.clear();
    ncomps++;
  }
  return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
  int n = sz(g);
  val.assign(n, 0); comp.assign(n, -1);
  Time = ncomps = 0;
  rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

BiconnectedComponents.h

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

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

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

2965e5, 33 lines

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

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

2sat.h

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

Usage:
```
TwoSat ts(number of boolean variables);
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.setValue(2); // Var 2 is true
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
ts.solve(); // Returns true iff it is solvable
ts.values[0..N-1] holds the assigned values to the vars
```

Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

5f9706, 56 lines

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

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

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

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

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

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

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

EulerWalk.h

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

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

780b64, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
  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]);
    if (it == end){ ret.push_back(x); s.pop_back(); continue; }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      eu[e] = 1; s.push_back(y);
    }}
  for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
  return {ret.rbegin(), ret.rend()};
}
```

7.5 Coloring

EdgeColoring.h

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

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

e210e2, 31 lines

```
vi edgeColoring(int N, vector<pii> eds) {
  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) {
    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)
      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) {
      int left = fan[i], right = fan[++i], e = cc[i];
      adj[u][e] = left;
      adj[left][e] = u;
      adj[right][e] = -1;
      free[right] = e;
    }
    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++);
  }
  rep(i,0,sz(eds))
    for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
  return ret;
}
```

7.6 Counting

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;
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];
  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;
  }
}
```

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.

f7c0bc, 49 lines

```
typedef vector<bitset<200>> vb;
struct Maxclique {
  double limit=0.025, pk=0;
  struct Vertex { int i, d=0; };
  typedef vector<Vertex> vv;
  vb e;
  vv V;
  vector<vi> C;
  vi qmax, q, S, old;
  void init(vv& r) {
    for (auto& v : r) v.d = 0;
    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;
  }
  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;
      q.push_back(R.back().i);
      vv T;
      for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
      if (sz(T)) {
        if (S[lev]++ / ++pk < limit) init(T);
        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]; };
          while (any_of(all(C[k]), f)) k++;
          if (k > mxk) mxk = k, C[mxk + 1].clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        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;
      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) {
    rep(i,0,sz(e)) V.push_back({i});
  }
};
```

Triangles.h

Description: Runs a callback on the triangles in a simple graph

6e4a69, 24 lines

```
template <class F>
void triangles(int V, const vector<pair<int, int>> &eds, F f) {
  vi st(V + 1, 0), ind(V, 0), to(eds.size()), eInd(eds.size());
  vector<int> &d = ind;
  for (auto &&e : eds) { d[e.first]++; d[e.second]++; }
  auto cmp = [&] (int v, int w) {
    return d[v] == d[w] ? v > w : d[v] > d[w];
  };
  for (auto &&e : eds)
    st[cmp(e.first, e.second) ? e.second : e.first]++;
  partial_sum(st.begin(), st.end(), st.begin());
  for (int i = 0, v, w; i < int(eds.size()); i++) {
    tie(v, w) = eds[i]; if (cmp(v, w)) swap(v, w);
    to[--st[v]] = w; eInd[st[v]] = i;
  }
  fill(ind.begin(), ind.end(), -1); for (int v = 0; v < V; v++)
        {
    rep(e1, st[v], st[v + 1]) ind[to[e1]] = eInd[e1];
    rep(e1, st[v], st[v + 1])
      for (int w = to[e1], e2 = st[w]; e2 < st[w + 1]; e2++)
        if (ind[to[e2]] != -1)
          f(v, w, to[e2], eInd[e1], eInd[e2], ind[to[e2]]);
    rep(e1, st[v], st[v + 1]) ind[to[e1]] = -1;
  }
}
```

7.7 Trees

LCA.h

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

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

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

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

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

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

CompressTree.h

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

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

"LCA.h" 9775a0, 21 lines

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

HLD.h

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

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

"../data-structures/LazySegmentTree.h" 6f34db, 46 lines

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

DirectedMST.h

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

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

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

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

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

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

ManhattanMST.h

Description: Computes the minimum spanning tree of a complete graph of N points where the edge weights is equal to the Manhattan distance between the points —xi - xj— + —yi - yj—. Generates up to 4N candidate edges with each point connected to its nearest neighbour in each octant.

e7b779, 28 lines

```
template <class T> struct ManhattanMST : public KruskalMST<T> {
  using Edge = typename KruskalMST<T>::Edge;
  static vector<Edge> generateCandidates(vector<pair<T, T>> P){
    vector<int> id(P.size()); iota(id.begin(), id.end(), 0);
    vector<Edge> ret; ret.reserve(P.size() * 4); rep(h, 0, 4) {
      sort(id.begin(), id.end(), [&] (int i, int j) {
        return P[i].first - P[j].first
            < P[j].second - P[i].second;
      });
      map<T, int> M; for (int i : id) {
        auto it = M.lower_bound(-P[i].second);
        for (; it != M.end(); it = M.erase(it)) {
          int j = it->second; T dx = P[i].first - P[j].first;
          T dy = P[i].second - P[j].second; if (dy > dx) break;
          ret.emplace_back(i, j, dx + dy);
        }
        M[-P[i].second] = i;
      }
      for (auto &&p : P) {
        if (h % 2) p.first = -p.first;
        else swap(p.first, p.second);
      }
    }
    return ret;
  }
  ManhattanMST(const vector<pair<T, T>> &P)
      : KruskalMST<T>(P.size(), generateCandidates(P)) {}
};
```

DominatorTree.h

Description: idom is a vector of the immediate dominator of each vertex, or -1 if it is not connected to the root, or is the root vertex.

c9ee81, 38 lines

```
struct DominatorTree {
  int V, ind; vector<int> idom, sdom, par, ord, U, UF, m;
  vector<vector<int>> bucket, H;
  int find(int v) {
    if (UF[v] == v) return v;
    int fv = find(UF[v]);
    if (sdom[m[v]] > sdom[m[UF[v]]]) m[v] = m[UF[v]];
    return UF[v] = fv;
  }
  int eval(int v) { find(v); return m[v]; }
  template <class Digraph> void dfs(const Digraph &G, int v) {
    ord[sdom[v] = ind++] = v; for (int w : G[v])
      if (sdom[w] == -1) { par[w] = v; dfs(G, w); }
  }
  template <class Digraph>
  DominatorTree(const Digraph &G, int root)
      : V(G.size()), ind(0), idom(V, -1), sdom(V, -1),
        par(V, -1), ord(V, -1), U(V, -1), UF(V), m(V),
        bucket(V), H(V) {
    for (int v = 0; v < V; v++) {
      UF[v] = m[v] = v; for (int w : G[v]) H[w].push_back(v);
    }
    dfs(G, root); for (int i = ind - 1; i > 0; i--) {
      int w = ord[i]; for (int v : H[w]) if (sdom[v] >= 0)
        sdom[w] = min(sdom[w], sdom[eval(v)]);
      bucket[ord[sdom[w]]].push_back(w);
      for (int v : bucket[par[w]]) U[v] = eval(v);
      bucket[UF[w] = par[w]].clear();
    }
    for (int i = 1; i < ind; i++) {
      int w = ord[i], u = U[w];
      idom[w] = sdom[w] == sdom[u] ? sdom[w] : idom[u];
    }
    for (int i = 1; i < ind; i++) {
      int w = ord[i]; idom[w] = ord[idom[w]];
    }
  }
};
```

7.8 Math

7.8.1 Number of Spanning Trees

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

7.8.2 Erdős–Gallai theorem

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

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

Geometry (8)

8.1 Primitives

EpsCmp.h

Description: Epsilon comparisons

a19131, 9 lines

```
using T = long double;
constexpr const T EPS = 1e-9;
bool lt(T a, T b) { return a + EPS < b; }
bool le(T a, T b) { return !lt(b, a); }
bool gt(T a, T b) { return lt(b, a); }
bool ge(T a, T b) { return !lt(a, b); }
bool eq(T a, T b) { return !lt(a, b) && !lt(b, a); }
bool ne(T a, T b) { return lt(a, b) || lt(b, a); }
int sgn(T a) { return lt(a, 0) ? -1 : lt(0, a) ? 1 : 0; }
```

Point.h

Description: Point operations

"EpsCmp.h" 6b8df4, 53 lines

```
#define OP(op, U, a, x, y) \
  pt operator op (U a) const { return pt(x, y); } \
  pt &operator op##= (U a) { return *this = *this op a; }
#define CMP(op, body) \
  bool operator op (pt p) const { return body; }
struct pt {
  T x, y; constexpr pt(T x = 0, T y = 0) : x(x), y(y) {}
  pt operator + () const { return *this; }
  pt operator - () const { return pt(-x, -y); }
  OP(+, pt, p, x + p.x, y + p.y) OP(-, pt, p, x - p.x, y - p.y)
  OP(*, T, a, x * a, y * a) OP(/, T, a, x / a, y / a)
  friend pt operator * (T a, pt p) {
    return pt(a * p.x, a * p.y);
  }
  bool operator < (pt p) const {
    return eq(x, p.x) ? lt(y, p.y) : lt(x, p.x);
  }
  CMP(<=, !(p < *this)) CMP(>, p < *this) CMP(>=, !(*this < p))
  CMP(==, !(*this < p) && !(p < *this))
  CMP(!=, *this < p || p < *this)
  OP(*, pt, p, x * p.x - y * p.y, y * p.x + x * p.y)
  OP(/, pt, p, (x * p.x + y * p.y) / (p.x * p.x + p.y * p.y),
               (y * p.x - x * p.y) / (p.x * p.x + p.y * p.y))
};
#undef OP
#undef CMP
istream &operator >> (istream &stream, pt &p) {
  return stream >> p.x >> p.y;
}
ostream &operator << (ostream &stream, pt p) {
  return stream << p.x << ' ' << p.y;
}
pt conj(pt a) { return pt(a.x, -a.y); }
T dot(pt a, pt b) { return a.x * b.x + a.y * b.y; }
T cross(pt a, pt b) { return a.x * b.y - a.y * b.x; }
T norm(pt a) { return dot(a, a); }
T abs(pt a) { return sqrt(norm(a)); }
T arg(pt a) { return atan2(a.y, a.x); }
pt polar(T r, T theta) {
  return r * pt(cos(theta), sin(theta));
}
T distSq(pt a, pt b) { return norm(b - a); }
T dist(pt a, pt b) { return abs(b - a); }
T ang(pt a, pt b) { return arg(b - a); }
T ang(pt a, pt b, pt c) {
  a -= b; c -= b; return arg(pt(dot(c, a), cross(c, a)));
}
T area2(pt a, pt b, pt c) { return cross(b - a, c - a); }
int ccw(pt a, pt b, pt c) { return sgn(area2(a, b, c)); }
pt rot(pt a, pt p, T theta) {
  return (a - p) * pt(polar(T(1), theta)) + p;
}
pt perp(pt a) { return pt(-a.y, a.x); }
```

Angle.h

Description: Angle wrapper for a point around a pivot

"Point.h" f5eb5f, 31 lines

```
#define OP(op, body) \
  Angle operator op (Angle a) const { return body; } \
  Angle &operator op##= (Angle a) {return *this = *this op a;}
#define CMP(op, body) \
  bool operator op (Angle a) const { return body; }
struct Angle {
  static pt pivot; static void setPivot(pt p) { pivot = p; }
  pt p; Angle(pt p = pt(0, 0)) : p(p) {}
  int half() const {
    if (eq(p.x, pivot.x) && eq(p.y, pivot.y)) return 2;
    return int(!lt(p.y, pivot.y)
        && (!eq(p.y, pivot.y) || !lt(p.x, pivot.x)));
  }
  bool operator < (Angle a) const {
    int h = half() - a.half();
    return h == 0 ? ccw(pivot, p, a.p) > 0 : h < 0;
  }
  CMP(<=, !(a < *this)) CMP(>, a < *this) CMP(>=, !(*this < a))
  CMP(==, !(*this < a) && !(a < *this))
  CMP(!=, *this < a || a < *this)
  Angle operator + () const { return *this; }
  Angle operator - () const {
    return Angle(pivot + conj(p - pivot));
  }
  OP(+, Angle(pivot + (p - pivot) * (a.p - pivot)))
  OP(-, *this + (-a))
};
#undef OP
#undef CMP

pt Angle::pivot = pt(0, 0);
```

Line.h

Description: Line operations

"Point.h" 718381, 85 lines

```
struct Line {
  pt v; T c;
  // ax + by = c, left side is ax + by > c
  Line(T a = 0, T b = 0, T c = 0) : v(b, -a), c(c) {}
  // direction vector v with offset c
  Line(pt v, T c) : v(v), c(c) {}
  // points p and q
  Line(pt p, pt q) : v(q - p), c(cross(v, p)) {}
  T eval(pt p) const { return cross(v, p) - c; }
  // sign of onLeft, dist: 1 if left of line, 0 if on line, -1
      if right of line
  int onLeft(pt p) const { return sgn(eval(p)); }
  T dist(pt p) const { return eval(p) / abs(v); }
  T distSq(pt p) const {T e = eval(p); return e * e / norm(v);}
  Line perpThrough(pt p) const { return Line(p, p + perp(v)); }
  Line translate(pt p) const {return Line(v, c + cross(v, p));}
  Line shiftLeft(T d) const { return Line(v, c + d * abs(v)); }
  pt proj(pt p) const {return p - perp(v) * eval(p) / norm(v);}
  pt refl(pt p) const {
    return p - perp(v) * T(2) * eval(p) / norm(v);
  }
  int cmpProj(pt p, pt q) const {
    return sgn(dot(v, p) - dot(v, q));
  }
};

Line bisector(Line l1, Line l2, bool interior) {
  T s = interior ? 1 : -1;
  return Line(l2.v / abs(l2.v) + l1.v / abs(l1.v) * s,
              l2.c / abs(l2.v) + l1.c / abs(l1.v) * s);
}

int lineLineIntersection(Line l1, Line l2, pt &res) {
  T d = cross(l1.v, l2.v);
  if (eq(d, 0)) return l2.v * l1.c == l1.v * l2.c ? 2 : 0;
  res = (l2.v * l1.c - l1.v * l2.c) / d; return 1;
}

bool onSeg(pt p, pt a, pt b) {
  return !ccw(p, a, b) && !lt(0, dot(a - p, b - p));
}

int segSegIntersects(pt a, pt b, pt p, pt q) {
  if (ccw(a, b, p) * ccw(a, b, q) < 0
      && ccw(p, q, a) * ccw(p, q, b) < 0)
    return 1;
  if (onSeg(p, a, b) || onSeg(q, a, b)
      || onSeg(a, p, q) || onSeg(b, p, q))
    return 2;
  return 0;
}

vector<pt> segSegIntersection(pt a, pt b, pt p, pt q) {
  int intersects = segSegIntersects(a, b, p, q);
  if (!intersects) return vector<pt>();
  if (intersects == 1) {
    T c1 = cross(p - a, b - a), c2 = cross(q - a, b - a);
    return vector<pt>{(c1 * q - c2 * p) / (c1 - c2)};
  }
  vector<pt> ret; if (onSeg(p, a, b)) ret.push_back(p);
  if (onSeg(q, a, b)) ret.push_back(q);
  if (onSeg(a, p, q)) ret.push_back(a);
  if (onSeg(b, p, q)) ret.push_back(b);
  sort(ret.begin(), ret.end());
  ret.erase(unique(ret.begin(), ret.end()), ret.end());
  return ret;
}

pt closestPtOnSeg(pt p, pt a, pt b) {
  if (a == b) return a;
  T d = distSq(a, b), t = min(d, max(T(0), dot(p - a, b - a)));
  return a + (b - a) * t / d;
}

T ptSegDist(pt p, pt a, pt b) {
  if (a == b) return dist(a, p);
  T d = distSq(a, b), t = min(d, max(T(0), dot(p - a, b - a)));
  return abs((p - a) * d - (b - a) * t) / d;
}

T segSegDist(pt a, pt b, pt p, pt q) {
  return segSegIntersects(a, b, p, q) > 0
      ? 0
      : min({ptSegDist(p, a, b), ptSegDist(q, a, b),
             ptSegDist(a, p, q), ptSegDist(b, p, q)});
}
```

Circle.h

Description: Opeartions with circles

"Point.h", "Angle.h", "Line.h" 6c0ddc, 155 lines

```
struct Circle {
  pt o; T r; Circle(T r = 0) : o(0, 0), r(r) {}
  Circle(pt o, T r) : o(o), r(r) {}
  int contains(pt p) const {return sgn(r * r - distSq(o, p));}
  int contains(Circle c) const {
    T dr = r - c.r;
    return lt(dr, 0) ? -1 : sgn(dr * dr - distSq(o, c.o));
  }
  int disjoint(Circle c) const {
    T sr = r + c.r; return sgn(distSq(o, c.o) - sr * sr);
  }
  pt proj(pt p) const { return o + (p - o) * r / dist(o, p); }
  pt inv(pt p) const { return o + (p - o)*r*r / distSq(o, p); }
};

vector<pt> circleLineIntersection(Circle c, Line l) {
  vector<pt> ret; T h2 = c.r * c.r - l.distSq(c.o);
  pt p = l.proj(c.o); if (eq(h2, 0)) ret.push_back(p);
  else if (lt(0, h2)) {
    pt h = l.v * sqrt(h2) / abs(l.v);
    ret.push_back(p - h); ret.push_back(p + h);
  }
  return ret;
}

vector<pt> circleSegIntersection(Circle c, pt a, pt b) {
  vector<pt> ret;
  if (a == b) { if (c.contains(a) == 0) ret.push_back(a); }
  else {
    Line l(a, b); for (auto &&p : circleLineIntersection(c, l))
      if (l.cmpProj(a, p) <= 0 && l.cmpProj(p, b) <= 0)
        ret.push_back(p);
  }
  return ret;
}

T circleHalfPlaneIntersectionArea(Circle c, Line l) {
  T h2 = c.r * c.r - l.distSq(c.o), ret = 0; if (!lt(h2, 0)) {
    pt p = l.proj(c.o), h = l.v * sqrt(max(h2, T(0)))/abs(l.v);
    pt a = p - h, b = p + h; T theta = abs(ang(a, c.o, b));
    ret = c.r * c.r * (theta - sin(theta)) / 2;
  }
  if (l.onLeft(c.o) > 0) ret = acos(T(-1)) * c.r * c.r - ret;
  return ret;
}

// first point is guaranteed to not be on the left side of
// line from c1.o to c2.o
int circleCircleIntersection(Circle c1, Circle c2,
                             vector<pt> &res) {
  pt d = c2.o - c1.o; T d2 = norm(d);
  if (eq(d2, 0)) return eq(c1.r, c2.r) ? 2 : 0;
  T pd = (d2 + c1.r * c1.r - c2.r * c2.r) / 2;
  T h2 = c1.r * c1.r - pd * pd / d2; pt p = c1.o + d * pd / d2;
  if (eq(h2, 0)) res.push_back(p);
  else if (lt(0, h2)) {
    pt h = perp(d) * sqrt(h2 / d2);
    res.push_back(p - h); res.push_back(p + h);
  }
  return !res.empty();
}

T circleCircleIntersectionArea(Circle c1, Circle c2) {
  T d = dist(c1.o, c2.o); if (!lt(d, c1.r + c2.r)) return 0;
  if (!lt(c2.r, d + c1.r)) return acos(T(-1)) * c1.r * c1.r;
  if (!lt(c1.r, d + c2.r)) return acos(T(-1)) * c2.r * c2.r;
  auto A = [&] (T r1, T r2) {
    T a = (d * d + r1 * r1 - r2 * r2) / (2 * d * r1);
    T theta = 2 * acos(max(T(-1), min(T(1), a)));
    return r1 * r1 * (theta - sin(theta)) / 2;
  };
  return A(c1.r, c2.r) + A(c2.r, c1.r);
}

// each pair represents a point on the first circle and the
// second circle; first point in each pair is guaranteed to not
// be on the left side of the line from c1.o to c2.o
int circleCircleTangent(Circle c1, Circle c2, bool inner,
                        vector<pair<pt, pt>> &res) {
  pt d = c2.o - c1.o; T r2 = inner ? -c2.r : c2.r;
  T dr = c1.r - r2, d2 = norm(d), h2 = d2 - dr * dr;
  if (eq(d2, 0) || lt(h2, 0)) return eq(h2, 0) ? 2 : 0;
  for (T sign : {T(-1), T(1)}) {
    pt v = (d * dr + perp(d) * sqrt(max(h2, T(0))) * sign)/d2;
    res.emplace_back(c1.o + v * c1.r, c2.o + v * r2);
  }
  return 1;
}

Circle circumcircle(pt a, pt b, pt c) {
  b -= a; c -= a;
  pt ret = b*c*(conj(c) - conj(b)) / (b*conj(c) - conj(b)*c);
  return Circle(a + ret, abs(ret));
}

template <class T, class Cmp = less<T>>
vector<pair<T, T>> &intervalUnion(vector<pair<T, T>> &A,
                                  Cmp cmp = Cmp()) {
  sort(A.begin(), A.end(), [&] (const pair<T, T> &a,
                                const pair<T, T> &b) {
    if (cmp(a.first, b.first)) return true;
    if (cmp(b.first, a.first)) return false;
    return cmp(a.second, b.second);
  });
  int i = 0;
  for (int l = 0, r = 0, N = A.size(); l < N; l = r, i++) {
    A[i] = A[l];
    for (r = l + 1; r < N && !cmp(A[i].second,A[r].first); r++)
      A[i].second = max(A[i].second, A[r].second, cmp);
  }
  A.erase(A.begin() + i, A.end()); return A;
}

T circleUnionArea(const vector<Circle> &circles) {
  int n = circles.size(); T ret = 0; rep(i, 0, n) {
    vector<pair<Angle, Angle>> intervals;
    Angle::setPivot(circles[i].o);
    bool inside = false; rep(j, 0, n) if (i != j) {
      int o = circles[j].contains(circles[i]);
      if (o > 0 || (o == 0 && (lt(circles[i].r, circles[j].r)
          || j < i))) {
        inside = true; break;
      }
      vector<pt> p;
      circleCircleIntersection(circles[i], circles[j], p);
      if (int(p.size()) == 2) {
        Angle a(p[0]), b(p[1]);
        if (a < b) intervals.emplace_back(a, b);
        else {
          intervals.emplace_back(a, Angle(circles[i].o));
          Angle c(circles[i].o - pt(circles[i].r, 0));
          intervals.emplace_back(c, b);
        }
      }
    }
    if (inside) continue;
    if (intervals.empty())
      ret += acos(T(-1)) * circles[i].r * circles[i].r;
    else {
      intervalUnion(intervals);
      if (intervals.back().second == circles[i].o) {
        intervals.front().first = intervals.back().first;
        intervals.pop_back();
      }
      for (int j = 0, k = int(intervals.size()); j < k; j++) {
        pt a = intervals[j].second.p;
        pt b = intervals[j + 1 == k ? 0 : j + 1].first.p;
        ret += cross(a, b) / 2;
        ret += circleHalfPlaneIntersectionArea(circles[i],
                                               Line(b, a));
      }
    }
  }
  return ret;
}
```

Polygon.h

Description: Polygon operations

"Point.h", "Angle.h", "Line.h", "Circle.h" e1e64e, 264 lines

```
int mod(int i, int n) { return i < n ? i : i - n; }

T getArea2(vector<pt> &poly) {
  T ret = 0; int n = poly.size();
  rep(i, 0, n) ret += cross(poly[i], poly[mod(i + 1, n)]);
  return ret;
}

pt getCentroid(vector<pt> &poly) {
  T A2 = 0; pt cen(0, 0); int n = poly.size(); rep(i, 0, n) {
    T a = cross(poly[i], poly[mod(i + 1, n)]); A2 += a;
    cen += a * (poly[i] + poly[mod(i + 1, n)]);
  }
  return cen / A2 / T(3);
}

int isCcwConvexPolygon(vector<pt> &poly) {
  return ccw(poly.back(), poly[0], poly[mod(1, poly.size())]);
}

int isCcwPolygon(vector<pt> &poly) {
  int n = poly.size();
  int i = min_element(poly.begin(), poly.end()) - poly.begin();
  return ccw(poly[mod(i+n-1, n)], poly[i], poly[mod(i+1, n)]);
}

int isInConvexPolygon(vector<pt> &poly, pt p) {
  int n = poly.size(), a = 1, b = n - 1;
  if (n < 3) return onSeg(p, poly[0], poly.back()) ? 0 : -1;
  if (ccw(poly[0], poly[a], poly[b]) > 0) swap(a, b);
  int o1 = ccw(poly[0],p,poly[a]), o2 = ccw(poly[0],p,poly[b]);
  if (o1 < 0 || o2 > 0) return -1;
  if (o1 == 0 || o2 == 0) return 0;
  while (abs(a - b) > 1) {
    int c = (a+b)/2; (ccw(poly[0],p,poly[c]) < 0 ? b : a) = c;
  }
  return ccw(poly[a], p, poly[b]);
}

int isInPolygon(vector<pt> &poly, pt p) {
  int n = poly.size(), windingNumber = 0; rep(i, 0, n) {
    pt a = poly[i], b = poly[mod(i + 1, n)];
    if (lt(b.y, a.y)) swap(a, b);
    if (onSeg(p, a, b)) return 0;
    windingNumber ^= (!lt(p.y,a.y)&&lt(p.y,b.y)&&ccw(p,a,b)>0);
  }
  return windingNumber == 0 ? -1 : 1;
}

// Finds a vertex that is the furthest point in that direction,
// selecting the rightmost vertex if there are multiple
int extremeVertex(vector<pt> &poly, pt dir) {
  int n = poly.size(), lo = 0, hi = n; pt pp = perp(dir);
  auto cmp = [&] (int i, int j) {
    return sgn(cross(pp, poly[mod(i, n)] - poly[mod(j, n)]));
  };
  auto extr = [&] (int i) {
    return cmp(i + 1, i) >= 0 && cmp(i, i + n - 1) < 0;
  };
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = lo + (hi - lo) / 2; if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

// (-1, -1) if no collision, (i, -1) if touching corner i,
// (i, i) if along side (i, i + 1), (i, j) if crossing sides
// (i, i + 1) and (j, j + 1) crossing corner i is treated as
// crossing side (i, i + 1)
// first index in pair is guaranteed to NOT be on the left side
pair<int, int> convexPolygonLineIntersection(
    vector<pt> &poly, Line l) {
  int n = poly.size();
  if (n == 1) return make_pair(l.onLeft(poly[0])==0?0:-1,-1);
  if (n == 2) {
    int o0 = l.onLeft(poly[0]), o1 = l.onLeft(poly[1]);
    if (o0 == 0 && o1 == 0) return make_pair(0, 0);
    if (o0 == 0) return make_pair(0, -1);
    if (o1 == 0) return make_pair(1, -1);
    return o0 == o1 ? make_pair(-1, -1) : make_pair(0, 1);
  }
  int endA = extremeVertex(poly, -perp(l.v));
  int endB = extremeVertex(poly, perp(l.v));
  auto cmpL = [&] (int i) { return l.onLeft(poly[i]); };
  pair<int, int> ret(-1, -1);
  if (cmpL(endA) > 0 || cmpL(endB) < 0) return ret;
  for (int i = 0; i < 2; i++) {
    int lo = endB, hi = endA; while (mod(lo + 1, n) != hi) {
      int m = mod((lo + hi + (lo < hi ? 0 : n)) / 2, n);
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    (i ? ret.second : ret.first) = mod(lo + !cmpL(hi), n);
    swap(endA, endB);
  }
  if (ret.first == ret.second) return make_pair(ret.first, -1);
  if (!cmpL(ret.first) && !cmpL(ret.second)) {
    switch ((ret.first - ret.second + n + 1) % n) {
      case 0: return make_pair(ret.first, ret.first);
      case 2: return make_pair(ret.second, ret.second);
    }
  }
  return ret;
}

// p is considered to be below polygon
int convexPolygonPointSingleTangent(
    vector<pt> &poly, pt p, bool left) {
  int n = poly.size(), o = ccw(p, poly[0], poly.back());
  bool farSide = o ? o < 0
      : lt(distSq(p, poly.back()), distSq(p, poly[0]));
  int lo = farSide != left, hi = lo + n - 2; while (lo <= hi) {
    int mid = lo + (hi - lo) / 2;
    if (ccw(p, poly[0], poly[mid]) == (left ? -1 : 1)) {
      if (farSide == left) hi = mid - 1;
      else lo = mid + 1;
    } else {
      if ((ccw(poly[mid], poly[mod(mid+1, n)], p) < 0) == left)
        hi = mid - 1;
      else lo = mid + 1;
    }
  }
  return mod(lo, n);
}

// c is considered to be below the polygon
pair<int, int> convexPolygonCircleTangent(
    vector<pt> &poly, Circle c, bool inner) {
  int n = poly.size(), a = 0, b = 0; vector<pair<pt, pt>> t;
  for (int h = 0; h < 2; h++) {
    pt q = t[h].second; int o = ccw(q, poly[0], poly.back());
    bool farSide = o ? o < 0
```

```
        : lt(distSq(q, poly.back()), distSq(q, poly[0]));
    int lo = farSide == h, hi = lo + n - 2; while (lo <= hi) {
      int mid = lo + (hi - lo) / 2; t.clear();
      q = t[h].second;
      if (ccw(q, poly[0], poly[mid]) == (h ? 1 : -1)) {
        if (farSide != h) hi = mid - 1;
        else lo = mid + 1;
      } else {
        if ((ccw(poly[mid], poly[mod(mid+1, n)], q) < 0) != h)
          hi = mid - 1;
        else lo = mid + 1;
      }
    }
    (h ? b : a) = mod(lo, n); t.clear();
  }
  return make_pair(a, b);
}

// second polygon is considered to be blow the first
vector<pair<int, int>> convexPolygonConvexPolygonTangent(
    vector<pt> &poly1, vector<pt> &poly2, bool inner) {
  int n = poly1.size(), a = 0, b = 0, c = 0, d = 0;
  vector<pair<int, int>> ret; for (int h = 0; h < 2; h++) {
    pt q = poly2[convexPolygonPointSingleTangent(
        poly2, poly1[0], inner ^ h)];
    int o = ccw(q, poly1[0], poly1.back());
    bool farSide = o ? o < 0
        : lt(distSq(q, poly1.back()), distSq(q, poly1[0]));
    int lo = farSide == h, hi = lo + n - 2; while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      q = poly2[convexPolygonPointSingleTangent(
          poly2, poly1[mid], inner ^ h)];
      if (ccw(q, poly1[0], poly1[mid]) == (h ? 1 : -1)) {
        if (farSide != h) hi = mid - 1;
        else lo = mid + 1;
      } else {
        if ((ccw(poly1[mid], poly1[mod(mid+1, n)],q) < 0) != h)
          hi = mid - 1;
        else lo = mid + 1;
      }
    }
    (h ? b : a) = lo = mod(lo, n);
    (h ? d : c) = convexPolygonPointSingleTangent(
        poly2, poly1[lo], inner ^ h);
  }
  ret.emplace_back(a, c); ret.emplace_back(b, d); return ret;
}

pt closestPointOnConvexPolygon(vector<pt> &poly, pt p) {
  pair<int, int> tangent = convexPolygonPointTangent(poly, p);
  int n = poly.size(), len = tangent.second - tangent.first;
  if (len < 0) len += n;
  if (len == 0) return poly[tangent.first];
  int lo = 0, hi = len - 2; while (lo <= hi) {
    int mid = lo+(hi-lo)/2, i = mod(tangent.first + mid, n);
    if (ptSegDist(p, poly[i], poly[mod(i + 1, n)])
        < ptSegDist(p, poly[mod(i+1, n)], poly[mod(i+2, n)]))
      hi = mid - 1;
    else lo = mid + 1;
  }
  int i = mod(tangent.first + lo, n);
  return closestPtOnSeg(p, poly[i], poly[mod(i + 1, n)]);
}

vector<pt> polygonHalfPlaneIntersection(
    vector<pt> &poly, Line l) {
  int n = poly.size(); vector<pt> ret; rep(i, 0, n) {
    int j = mod(i + 1, n);
    int o1 = l.onLeft(poly[i]), o2 = l.onLeft(poly[j]);
    if (o1 >= 0) ret.push_back(poly[i]);
    if (o1 && o2 && o1 != o2) {
      pt p; if (lineLineIntersection(
          l, Line(poly[i], poly[j]), p) == 1)
        ret.push_back(p);
    }
  }
  return ret;
}

T polygonUnion(vector<vector<pt>> &polys) {
  auto rat = [&] (pt p, pt q) {
    return sgn(q.x) ? p.x / q.x : p.y / q.y;
  };
  T ret = 0; for (int i = 0; i < int(polys.size()); i++)
    for (int v = 0; v < int(polys[i].size()); v++) {
      pt a=polys[i][v], b=polys[i][mod(v+1,polys[i].size())];
      vector<pair<T, int>> segs{
          make_pair(0, 0), make_pair(1, 0)};
      for (int j = 0; j < int(polys.size()); j++) if (i != j)
        for (int w = 0; w < int(polys[j].size()); w++) {
          pt c = polys[j][w];
          pt d = polys[j][mod(w + 1, polys[j].size())];
          int sc = ccw(a,b,c), sd = ccw(a,b,d); if (sc != sd) {
            if (min(sc, sd) < 0) {
              T sa = area2(c, d, a), sb = area2(c, d, b);
              segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
            }
          } else if (j<i && !sc && !sd && sgn(dot(b-a,d-c))>0){
            segs.emplace_back(rat(c - a, b - a), 1);
            segs.emplace_back(rat(d - a, b - a), -1);
          }
        }
      sort(segs.begin(), segs.end()); T sm = 0;
      for (auto &&s : segs)s.first=min(max(s.first,T(0)),T(1));
      for (int j = 1, cnt = segs[0].second;
          j < int(segs.size()); j++) {
        if (!cnt) sm += segs[j].first - segs[j - 1].first;
        cnt += segs[j].second;
      }
      ret += cross(a, b) * sm;
    }
  return ret / 2;
}

T polygonCircleIntersectionArea(vector<pt> &poly, Circle c) {
  T r2 = c.r * c.r / 2;
  auto tri = [&] (pt p, pt q) {
    pt d = q - p; T a = dot(d, p) / norm(d);
    T b = (norm(p) - c.r * c.r) / norm(d), det = a * a - b;
    if (!lt(0, det)) return ang(q, pt(0, 0), p) * r2;
    T s = max(T(0), -a-sqrt(det)), t = min(T(1), -a+sqrt(det));
    if (lt(t, 0) || !lt(s, 1)) return ang(q, pt(0, 0), p) * r2;
    pt u = p + d * s, v = p + d * t;
    return ang(u, pt(0, 0), p) * r2 + cross(u, v) / 2
        + ang(q, pt(0, 0), v) * r2;
  };
  T ret = 0; for (int n = poly.size(), i = 0; i < n; i++)
    ret += tri(poly[i] - c.o, poly[mod(i + 1, n)] - c.o);
  return ret;
}
```

8.2 Algorithms

ConvexHull.h

Description: Convex hull

"Point.h" 2ed3ea, 15 lines

```
vector<pt> convexHull(vector<pt> P) {
  vector<pt> hull; sort(P.begin(), P.end()); rep(h, 0, 2) {
    int st = hull.size(); for (auto &&p : P) {
      while (int(hull.size()) >= st + 2
          && ccw(hull[hull.size() - 2], hull.back(), p) <= 0)
        hull.pop_back();
      hull.push_back(p);
    }
    hull.pop_back(); reverse(P.begin(), P.end());
  }
  if (int(hull.size()) == 2 && hull[0] == hull[1])
    hull.pop_back();
  if (hull.empty() && !P.empty()) hull.push_back(P[0]);
  return hull;
}
```

ClosestPair.h

Description: Closest pair of points

"Point.h" 881b04, 29 lines

```
struct ClosestPair {
  static bool yOrdLt(pt p, pt q) { return lt(p.y, q.y); }
  pt best1, best2; T bestDistSq;
  void closest(vector<pt> &P, int lo, int hi) {
    if (hi <= lo) return;
    int mid = lo + (hi - lo) / 2; pt median = P[mid];
    closest(P, lo, mid);closest(P, mid + 1, hi);vector<pt> aux;
    merge(P.begin()+lo, P.begin()+mid+1, P.begin()+mid+1,
          P.begin() + hi + 1, back_inserter(aux), yOrdLt);
    copy(aux.begin(), aux.end(), P.begin() + lo);
    for (int i = lo, k = 0; i <= hi; i++) {
      T dx = P[i].x - median.x, dx2 = dx * dx;
      if (lt(dx2, bestDistSq)) {
        for (int j = k - 1; j >= 0; j--) {
          T dy = P[i].y - aux[j].y, dy2 = dy * dy;
          if (!lt(dy2, bestDistSq)) break;
          T dSq = distSq(P[i],aux[j]); if (lt(dSq,bestDistSq)){
            bestDistSq = dSq; best1 = P[i]; best2 = aux[j];
          }
        }
        aux[k++] = P[i];
      }
    }
  }
  ClosestPair(vector<pt> P)
      : bestDistSq(numeric_limits<T>::max()) {
    sort(P.begin(), P.end()); closest(P, 0, int(P.size()) - 1);
  }
};
```

FarthestPair.h

Description: Farthest pair of points

"Point.h", "ConvexHull.h" 495948, 16 lines

```
struct FarthestPair {
  pt best1, best2; T bestDistSq; vector<pt> hull;
  FarthestPair(const vector<pt> &P)
      : bestDistSq(0), hull(convexHull(P)) {
    int H = hull.size(); pt o(0, 0);
    for (int i = 0, j = H < 2 ? 0 : 1; i < j; i++)
      for (;; j = (j + 1) % H) {
        T dSq = distSq(hull[i], hull[j]);
        if (lt(bestDistSq, dSq)) {
          bestDistSq = dSq; best1 = hull[i]; best2 = hull[j];
        }
        pt a = hull[i+1]-hull[i], b = hull[(j+1)%H]-hull[j];
        if (ccw(o, a, b) <= 0) break;
      }
  }
};
```

MinEnclosingCircle.h

Description: Finds the minimum enclosing circle of a set of points

"Point.h", "Circle.h" 54d53a, 13 lines

```
Circle minEnclosingCircle(vector<pt> P) {
  mt19937_64 rng(0); shuffle(P.begin(), P.end(), rng);
  Circle c(P[0], 0);
  rep(i, 0, P.size()) if (lt(c.r, dist(P[i], c.o))) {
    c = Circle(P[i], 0);
    rep(j, 0, i) if (lt(c.r, dist(P[j], c.o))) {
      pt p = (P[i] + P[j])/T(2); c = Circle(p, dist(P[i], p));
      rep(k, 0, j) if (lt(c.r, dist(P[k], c.o)))
        c = circumcircle(P[i], P[j], P[k]);
    }
  }
  return c;
}
```

HalfPlaneIntersection.h

Description: Computes the intersection of half-planes defined by the left side of a set of lines (including the line itself). Code 0 = finite intersection, 1 = infinite and bounded by line segs and rays, 2 = infinite and bounded by two lines (4 points given), 3 = entire plane

"Point.h", "Angle.h", "Line.h" a03aa6, 51 lines

```
pair<int,vector<pt>> halfPlaneIntersection(vector<Line> lines){
  Angle::setPivot(pt(0, 0));
  sort(lines.begin(), lines.end(), [&] (Line a, Line b) {
    Angle angA(a.v), angB(b.v);
    return angA == angB ? a.onLeft(b.proj(pt(0, 0))) < 0
      : angA < angB;
  });
  lines.erase(unique(lines.begin(), lines.end(),
      [&] (Line a, Line b) {
    return Angle(a.v) == Angle(b.v);
  }), lines.end());
  int N = lines.size();
  if (N == 0) return make_pair(3, vector<pt>());
  if (N == 1) {
    pt p = lines[0].proj(pt(0, 0));
    return make_pair(1, vector<pt>{p, p + lines[0].v});
  }
  int code = 0; for (int i = 0; code == 0 && i < N; i++) {
    Angle diff=Angle(lines[i].v)-Angle(lines[i==0?N-1:i-1].v);
    if (diff < Angle(pt(1, 0))) {
      rotate(lines.begin(), lines.begin() + i, lines.end());
      code = 1; if (N == 2 && diff == Angle(pt(-1, 0))) {
        pt p = lines[0].proj(pt(0, 0));
        if (lines[1].onLeft(p) < 0)
          return make_pair(0, vector<pt>());
        pt q = lines[1].proj(pt(0, 0));
        return make_pair(2, vector<pt>{p, p + lines[0].v,
            q, q + lines[1].v});
      }
    }
  }
  vector<Line> q(N + 1, lines[0]); vector<pt> ret(N); pt inter;
  int front = 0, back = 0; for (int i = 1; i <= N - code; i++)
      {
    if (i == N) lines.push_back(q[front]);
    while (front < back && lines[i].onLeft(ret[back - 1]) < 0)
      back--;
    while (i != N && front < back
        && lines[i].onLeft(ret[front]) < 0) front++;
    if (lineLineIntersection(lines[i], q[back], inter) != 1)
      continue;
    ret[back++] = inter; q[back] = lines[i];
  }
  if (code == 0 && back - front < 3)
    return make_pair(code, vector<pt>());
  vector<pt> P(ret.begin() + front, ret.begin() + back);
  if (code == 1) {
    P.insert(P.begin(), P[0] - q[front].v);
    P.push_back(P.back() + q[back].v);
  }
  return make_pair(code, P);
}
```

DelaunayTriangulation.h

Description: If all points are collinear, there is no triangulation. If there are 4 or more points on the same circle, the triangulation is ambiguous, otherwise it is unique. Each circumcircle does not completely contain any of the input points.

"Point.h" dd0470, 103 lines

```
struct DelaunayTriangulation {
  static constexpr const pt inf = pt(numeric_limits<T>::max(),
                                     numeric_limits<T>::max());
  struct Quad {
    static vector<Quad> V;
    int rot, o; pt p; Quad(int rot) : rot(rot), o(-1), p(inf){}
    int &r() { return V[rot].rot; }
    pt &f() { return V[r()].p; }
    int prv() { return V[V[rot].o].rot; }
    int nxt() { return V[r()].prv(); }
  };
  int head; vector<Quad> &V = Quad::V; vector<array<pt,3>> tri;
  int makeQuad(int rot) {
    V.emplace_back(rot); return int(V.size()) - 1;
  }
  bool circ(pt p, pt a, pt b, pt c) {
    T p2 = norm(p), A = norm(a) - p2;
    T B = norm(b) - p2C = norm(c) - p2;
    return lt(0, area2(p, a, b) * C + area2(p, b, c) * A
        + area2(p, c, a) * B);
  }
  int makeEdge(pt a, pt b) {
    int r = ~head ? head
        : makeQuad(makeQuad(makeQuad(makeQuad(-1))));
    head = V[r].o; V[V[r].r()].r() = r; rep(i, 0, 4) {
      r = V[r].rot; V[r].o = i & 1 ? r : V[r].r();
    }
    V[r].p = a; V[r].f() = b; return r;
  }
  void splice(int a, int b) {
    swap(V[V[V[a].o].rot].o, V[V[V[b].o].rot].o);
    swap(V[a].o, V[b].o);
  }
  int connect(int a, int b) {
    int q = makeEdge(V[a].f(), V[b].p);
    splice(q, V[a].nxt()); splice(V[q].r(), b); return q;
  }
  bool valid(int a, int base) {
    return ccw(V[a].f(), V[base].f(), V[base].p) > 0;
  }
  pair<int, int> rec(const vector<pt> &P, int lo, int hi) {
    int k = hi - lo + 1; if (k <= 3) {
      int a = makeEdge(P[lo], P[lo + 1]);
      int b = makeEdge(P[lo + 1], P[hi]);
      if (k == 2) return make_pair(a, V[a].r());
      splice(V[a].r(), b);
      int side = sgn(ccw(P[lo], P[lo + 1], P[hi]));
      int c = side ? connect(b, a) : -1;
      return make_pair(side < 0 ? V[c].r() : a,
                       side < 0 ? c : V[b].r());
    }
    int a, b, ra, rb, mid = lo + (hi - lo) / 2;
    tie(ra, a) = rec(P, lo, mid);
    tie(b, rb) = rec(P, mid + 1, hi);
    while ((ccw(V[b].p,V[a].f(),V[a].p)<0&&(a=V[a].nxt()))
        || (ccw(V[a].p,V[b].f(),V[b].p)>0&&(b=V[V[b].r()].o)));
    int base = connect(V[b].r(), a);
    if (V[a].p == V[ra].p) ra = V[base].r();
    if (V[b].p == V[rb].p) rb = base;
    while (true) {
      int l = V[V[base].r()].o; if (valid(l, base))
        while (circ(V[V[l].o].f(), V[base].f(),
               V[base].p, V[l].f())) {
          int t = V[l].o; splice(l, V[l].prv());
          splice(V[l].r(), V[V[l].r()].prv()); V[l].o = head;
          head = l; l = t;
        }
      int r = V[base].prv(); if (valid(r, base))
        while (circ(V[V[r].prv()].f(), V[base].f(),
               V[base].p, V[r].f())) {
          int t = V[r].prv(); splice(r, V[r].prv());
          splice(V[r].r(), V[V[r].r()].prv()); V[r].o = head;
          head = r; r = t;
        }
      if (!valid(l, base) && !valid(r, base)) break;
      if (!valid(l, base) || (valid(r, base)
          && circ(V[r].f(), V[r].p, V[l].f(), V[l].p)))
        base = connect(r, V[base].r());
      else base = connect(V[base].r(), V[l].r());
    }
    return make_pair(ra, rb);
  }
  DelaunayTriangulation(vector<pt> P) : head(-1) {
    sort(P.begin(), P.end()); V.reserve(P.size() * 16);
    if (int(P.size()) < 2) return;
    int e = rec(P, 0, int(P.size())-1).first, qi = 0, ind = 0;
    vector<bool> vis(V.size(), false);
    vector<int> q{e}; q.reserve(V.size());
    while (ccw(V[V[e].o].f(),V[e].f(),V[e].p) < 0) e = V[e].o;
    auto add = [&] {
      int c = e; do {
        if (ind % 3 == 0) tri.emplace_back();
        tri.back()[ind++ % 3] = V[c].p;
        vis[c] = true; q.push_back(V[c].r()); c = V[c].nxt();
      } while (c != e);
    };
    add(); tri.clear(); ind = 0; tri.reserve(V.size() / 3 + 1);
    while (qi < int(q.size())) if (!vis[e = q[qi++]]) add();
    assert(ind % 3 == 0); V = vector<Quad>();
  }
};

vector<DelaunayTriangulation::Quad> DelaunayTriangulation::Quad
     ::V = vector<DelaunayTriangulation::Quad>();
```

VoronoiDiagram.h

Description: The Voronoi Diagram is represented as a graph with the first tri.size() vertices being the circumcenters of each triangle in the triangulation and additional vertices being added to represent infinty points in a certain direction. An edge is added between (v, w) if either triangles v and w share an edge, or if v is a triangle that has an edge that is not shared with any other triangle and rayDir[w - tri.size()] is the direction of the ray from the circumcenter of v passing perpendicular to the edge of the triangle of v that is not shared.

"Point.h", "Circle.h", "DelaunayTriangulation.h" b25035, 25 lines

```
struct VoronoiDiagram {
  vector<array<pt, 3>> tri; vector<Circle> circumcircles;
  vector<pt> rayDir; vector<vector<int>> G;
  VoronoiDiagram(const vector<pt> &P)
      : tri(DelaunayTriangulation(P).tri), G(tri.size()) {
    map<pair<pt, pt>, int> seen;
    circumcircles.reserve(tri.size()); for (auto &&t : tri)
```

```
      circumcircles.push_back(circumcircle(t[0], t[1], t[2]));
    rep(h, 0, 2) rep(i, 0, tri.size()) rep(k, 0, 3) {
      pt p1 = tri[i][k], p2 = tri[i][(k + 1) % 3];
      auto it = seen.find(make_pair(p1, p2)); if (h == 0) {
        if (it == seen.end()) it = seen.find(make_pair(p2,p1));
        if (it == seen.end()) seen[make_pair(p1, p2)] = i;
        else {
          int j = it->second;
          G[i].push_back(j); G[j].push_back(i); seen.erase(it);
        }
      } else if (it != seen.end()) {
        int j = G.size(); G.emplace_back();
        rayDir.push_back(perp(p1 - p2));
        G[i].push_back(j); G[j].push_back(i);
      }
    }
  }
};
```

8.3 3D

Point3D.h

Description: Point operations in 3D

"../../utils/EpsCmp.h" b199ad, 69 lines

```
#define OP(op, U, a, x, y, z) \
  pt3 operator op (U a) const { return pt3(x, y, z); } \
  pt3 &operator op##= (U a) { return *this = *this op a; }
#define CMP(op, body) \
  bool operator op (pt3 p) const { return body; }
struct pt3 {
  T x, y, z;
  constexpr pt3(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  pt3 operator + () const { return *this; }
  pt3 operator - () const { return pt3(-x, -y, -z); }
  OP(+, pt3, p, x + p.x, y + p.y, z + p.z)
  OP(-, pt3, p, x - p.x, y - p.y, z - p.z)
  OP(*, T, a, x * a, y * a, z * a)
  OP(/, T, a, x / a, y / a, z / a)
  friend pt3 operator * (T a, pt3 p) {
    return pt3(a * p.x, a * p.y, a * p.z);
  }
  bool operator < (pt3 p) const {
    return eq(x, p.x) ? (eq(y, p.y)
        ? lt(z, p.z) : lt(y, p.y)) : lt(x, p.x);
  }
  CMP(<=, !(p < *this)) CMP(>, p < *this) CMP(>=, !(*this < p))
  CMP(==, !(*this < p) && !(p < *this))
  CMP(!=, *this < p || p < *this)
  T operator | (pt3 p) const {
    return x * p.x + y * p.y + z * p.z;
  }
  OP(*, pt3, p, y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x)
};
#undef OP
#undef CMP
istream &operator >> (istream &stream, pt3 &p) {
  return stream >> p.x >> p.y >> p.z;
}
ostream &operator << (ostream &stream, pt3 p) {
  return stream << p.x << ' ' << p.y << ' ' << p.z;
}
T norm(pt3 p) { return p | p; }
T abs(pt3 p) { return sqrt(norm(p)); }
pt3 unit(pt3 p) { return p / abs(p); }
T distSq(pt3 a, pt3 b) { return norm(b - a); }
T dist(pt3 a, pt3 b) { return abs(b - a); }
// returns an angle in the range [0, PI]
T ang(pt3 a, pt3 b, pt3 c) {
  a = unit(a - b); c = unit(c - b);
  return 2 * atan2(abs(a - c), abs(a + c));
}
pt3 rot(pt3 a, pt3 axis, T theta) {
  return a * cos(theta) + (unit(axis) * a * sin(theta))
      + (unit(axis) * (unit(axis) | a) * (1 - cos(theta)));
}
T volume6(pt3 a, pt3 b, pt3 c, pt3 d) {
  return (b - a) * (c - a) | (d - a);
}
int above(pt3 a, pt3 b, pt3 c, pt3 d) {
  return sgn(volume6(a, b, c, d));
}
// Converts a position based on radius (r >= 0),
// inclination/latitude (-PI / 2 <= theta <= PI / 2),
// and azimuth/longitude (-PI < phi <= PI)
// Convention is that the x axis passes through the meridian
// (phi = 0), and the z axis passes through the North Pole
// (theta = Pi / 2)
pt3 sph(T r, T theta, T phi) {
  return pt3(r * cos(theta) * cos(phi),
             r * cos(theta) * sin(phi), r * sin(theta));
}
T inc(pt3 p) { return atan2(p.z, T(sqrt(p.x*p.x + p.y*p.y))); }
T az(pt3 p) { return atan2(p.y, p.x); }
```

Line3D.h

Description: Line operations in 3D

"Point3D.h" 2bdb99, 81 lines

```
// represented in parametric form o + kd for real k
struct Line3D {
  pt3 o, d;
  Line3D(pt3 p=pt3(0,0,0), pt3 q=pt3(0,0,0)) : o(p), d(q-p) {}
  bool onLine(pt3 p) const { return eq(norm(d * (p - o)), 0); }
  T distSq(pt3 p) const { return norm(d * (p - o)) / norm(d); }
  T dist(pt3 p) const { return sqrt(distSq(p)); }
  Line3D translate(pt3 p) const { return Line3D(o + p, d); }
  pt3 proj(pt3 p) const { return o+d*(d|(p-o))/norm(d); }
  pt3 refl(pt3 p) const { return proj(p) * T(2) - p; }
  int cmpProj(pt3 p, pt3 q) const { return sgn((d|p)-(d|q)); }
};

T lineLineDist(Line3D l1, Line3D l2) {
  pt3 n = l1.d*l2.d; if (eq(norm(n), 0)) return l1.dist(l2.o);
  return abs((l2.o - l1.o) | n) / abs(n);
}

pt3 closestOnL1ToL2(Line3D l1, Line3D l2) {
  pt3 n = l1.d * l2.d;
  if (eq(norm(n), 0)) return l1.proj(pt3(0, 0, 0));
  pt3 n2 = l2.d * n;
  return l1.o + l1.d * ((l2.o - l1.o) | n2) / (l1.d | n2);
}

int lineLineIntersection(Line3D l1, Line3D l2, pt3 &res) {
  pt3 n = l1.d * l2.d;
  if (eq(norm(n), 0)) return eq(l1.dist(l2.o), 0) ? 2 : 0;
  res = closestOnL1ToL2(l1, l2); return 1;
}

bool onSeg(pt3 p, pt3 a, pt3 b) {
  if (a == b) return p == a;
  Line3D l(a, b);
  return l.onLine(p) && l.cmpProj(a, p) <= 0
      && l.cmpProj(p, b) <= 0;
}

vector<pt3> segSegIntersection(pt3 a, pt3 b, pt3 p, pt3 q) {
  vector<pt3> ret; if (a == b) {
    if (onSeg(a, p, q)) ret.push_back(a);
  } else if (p == q) {
    if (onSeg(p, a, b)) ret.push_back(p);
  } else {
    pt3 inter; Line3D l1(a, b), l2(p, q);
    int cnt = lineLineIntersection(l1, l2, inter);
    if (cnt == 1) ret.push_back(inter);
    else if (cnt == 2) {
      if (onSeg(p, a, b)) ret.push_back(p);
      if (onSeg(q, a, b)) ret.push_back(q);
      if (onSeg(a, p, q)) ret.push_back(a);
      if (onSeg(b, p, q)) ret.push_back(b);
    }
  }
  sort(ret.begin(), ret.end());
  ret.erase(unique(ret.begin(), ret.end()), ret.end());
  return ret;
}

pt3 closestPtOnSeg(pt3 p, pt3 a, pt3 b) {
  if (a != b) {
    Line3D l(a, b);
    if (l.cmpProj(a,p)<0 && l.cmpProj(p,b)<0) return l.proj(p);
  }
  return lt(dist(p, a), dist(p, b)) ? a : b;
}

T ptSegDist(pt3 p, pt3 a, pt3 b) {
  if (a != b) {
    Line3D l(a, b);
    if (l.cmpProj(a,p)<0 && l.cmpProj(p,b)<0) return l.dist(p);
  }
  return min(dist(p, a), dist(p, b));
}

T segSegDist(pt3 a, pt3 b, pt3 p, pt3 q) {
  return !segSegIntersection(a, b, p, q).empty()
      ? 0
      : min({ptSegDist(p, a, b), ptSegDist(q, a, b),
             ptSegDist(a, p, q), ptSegDist(b, p, q)});
}
```

Plane3D.h

Description: Plane operations in 3D

"Point3D.h", "Line3D.h" d59a2b, 68 lines

```
using namespace std;

struct Plane3D {
  pt3 n; T d;
  // ax + by + cz = d, above is ax + by + cz > d
  Plane3D(T a=0, T b=0, T c=0, T d=0) : n(a, b, c), d(d) {}
  // normal n, offset d
  Plane3D(pt3 n, T d) : n(n), d(d) {}
  // normal n, point p
  Plane3D(pt3 n, pt3 p) : n(n), d(n | p) {}
  // 3 non-collinear points p, q, r
  Plane3D(pt3 p, pt3 q, pt3 r) : Plane3D((q - p)*(r - p), p) {}
  T eval(pt3 p) const { return (n | p) - d; }
  int isAbove(pt3 p) const { return sgn(eval(p)); }
  T dist(pt3 p) const { return eval(p) / abs(n); }
  T distSq(pt3 p) const { T e = eval(p); return e*e/norm(n); }
  Plane3D translate(pt3 p) const {return Plane3D(n, d+(n|p));}
  Plane3D shiftUp(T e) const { return Plane3D(n, d+e*abs(n)); }
  pt3 proj(pt3 p) const { return p - n * eval(p) / norm(n); }
  pt3 refl(pt3 p) const { return p - n*T(2)*eval(p)/norm(n); }
  // looking from above
  int ccwProj(pt3 a, pt3 b, pt3 c) const {
    return sgn((b - a) * (c - a) | n);
  }
  // tuple (a, b, c) of 3 non-collinear points on the plane
  tuple<pt3, pt3, pt3> getPts() const {
    pt3 v = pt3(1,0,0); if (eq(abs(n * v), 0)) v = pt3(0,1,0);
    pt3 v1 = n * v, v2 = n * v1; pt3 a = proj(pt3(0,0,0));
    return make_tuple(a, a + v1, a + v2);
  }
};
Line3D perpThrough(Plane3D pi, pt3 o) {
  Line3D ret; ret.o = o; ret.d = pi.n; return ret;
}
Plane3D perpThrough(Line3D l, pt3 o) {return Plane3D(l.d, o);}

// Transforms points to a new coordinate system where the x and
// y axes are on the plane, with the z axis being the normal
// vector (positive z is in the direction of the normal vector)
// Z coordinate is guaranteed to be the distance to the plane
// (positive if above plane, negative if below, 0 if on)
struct CoordinateTransformation {
  pt3 o, dx, dy, dz;
  CoordinateTransformation(Plane3D pi) {
    pt3 p, q, r; tie(p, q, r) = pi.getPts(); o = p;
    dx = unit(q - p); dz = unit(dx * (r - p)); dy = dz * dx;
  }
  CoordinateTransformation(pt3 p, pt3 q, pt3 r) : o(p) {
    dx = unit(q - p); dz = unit(dx * (r - p)); dy = dz * dx;
  }
  pt3 transform(pt3 p) const {
    return pt3((p - o) | dx, (p - o) | dy, (p - o) | dz);
  }
};

int planeLineIntersection(Plane3D pi, Line3D l, pt3 &res) {
  T a = pi.n | l.d;
  if (eq(norm(a), 0)) return pi.isAbove(l.o) == 0 ? 2 : 0;
  res = l.o - l.d * pi.eval(l.o) / a; return 1;
}

int planePlaneIntersection(Plane3D pi1, Plane3D pi2, Line3D &
     res) {
  pt3 d = pi1.n * pi2.n; if (eq(norm(d), 0))
    return eq(abs(pi1.d / abs(pi1.n)), abs(pi2.d / abs(pi2.n)))
        ? 2 : 0;
  res.o = (pi2.n * pi1.d - pi1.n * pi2.d) * d / norm(d);
  res.d = d; return 1;
}
```

Sphere3D.h

Description: Sphere operations

"Point3D.h", "Line3D.h", "Plane3D.h" a5982e, 111 lines

```
struct Sphere3D {
  pt3 o; T r; Sphere3D(T r = 0) : o(0, 0), r(r) {}
  Sphere3D(pt3 o, T r) : o(o), r(r) {}
  int contains(pt3 p) const {return sgn(r * r - distSq(o, p));}
  int contains(Sphere3D s) const {
    T dr = r - s.r;
    return lt(dr, 0) ? -1 : sgn(dr * dr - distSq(o, s.o));
  }
  int disjoint(Sphere3D s) const {
    T sr = r + s.r; return sgn(distSq(o, s.o) - sr * sr);
  }
  pt3 proj(pt3 p) const { return o + (p - o) * r / dist(o, p);
      }
  pt3 inv(pt3 p) const { return o + (p - o)*r*r/distSq(o, p); }
  T greatCircleDist(pt3 a, pt3 b) const {return r*ang(a,o,b);}
  bool isGreatCircleSeg(pt3 a, pt3 b) const {
    assert(contains(a) == 0 && contains(b) == 0);
    a -= o; b -= o; return !eq(norm(a*b), 0) || lt(0, (a|b));
  }
  bool onSphSeg(pt3 p, pt3 a, pt3 b) const {
    p -= o; a -= o; b -= o; pt3 n = a * b;
    if (eq(norm(n), 0)) return eq(norm(a*p),0) && lt(0,(a|p));
    return eq((n|p),0) && !lt((n|a*p),0) && !lt(0,(n|b*p));
  }
  vector<pt3> greatCircleSegIntersection(pt3 a, pt3 b,
      pt3 p, pt3 q) const {
    pt3 ab = (a - o) * (b - o), pq = (p - o) * (q - o);
    int oa = sgn(pq | (a - o)), ob = sgn(pq | (b - o));
    int op = sgn(ab | (p - o)), oq = sgn(ab | (q - o));
    if (oa != ob && op != oq && oa != op)
      return vector<pt3>{proj(o + ab * pq * op)};
    vector<pt3> ret; if (onSphSeg(p, a, b)) ret.push_back(p);
    if (onSphSeg(q, a, b)) ret.push_back(q);
    if (onSphSeg(a, p, q)) ret.push_back(a);
    if (onSphSeg(b, p, q)) ret.push_back(b);
    sort(ret.begin(), ret.end());
    ret.erase(unique(ret.begin(), ret.end()), ret.end());
    return ret;
  }
  T angSph(pt3 a, pt3 b, pt3 c) const {
    a -= o; b -= o; c -= o;
    T theta = ang(b * a, pt3(0, 0, 0), b * c);
    return (a * b | c) < 0 ? -theta : theta;
  }
  T surfaceAreaOnSph(const vector<pt3> &poly) const {
    int n = poly.size(); T PI = acos(T(-1)), a = -(n - 2) * PI;
    for (int i = 0; i < n; i++) {
      T ang = angSph(poly[i], poly[(i+1)%n], poly[(i+2)%n]);
      if (ang < 0) ang += 2 * PI;
      a += ang;
    }
    return r * r * a;
  }
};

vector<pt3> sphereLineIntersection(Sphere3D s, Line3D l) {
  vector<pt3> ret; T h2 = s.r * s.r - l.distSq(s.o);
  pt3 p = l.proj(s.o); if (eq(h2, 0)) ret.push_back(p);
  else if (lt(0, h2)) {
    pt3 h = l.d * sqrt(h2) / abs(l.d);
    ret.push_back(p - h); ret.push_back(p + h);
  }
  return ret;
}

// res contains circle centre and radius
bool spherePlaneIntersection(
    Sphere3D s, Plane3D pi, pair<pt3, T> &res) {
  T d2 = s.r * s.r - pi.distSq(s.o);
  if (lt(d2, 0)) return false;
  res.first = pi.proj(s.o); res.second = sqrt(max(d2, T(0)));
  return true;
}

pair<T, T> sphereHalfSpaceIntersectionSAV(
    Sphere3D s, Plane3D pi) {
  T d2 = s.r * s.r - pi.distSq(s.o);
  T h = lt(d2, 0) ? T(0) : s.r - abs(pi.dist(s.o));
  if (pi.isAbove(s.o) > 0) h = s.r * 2 - h;
  T PI = acos(T(-1));
  return make_pair(PI*2*s.r*h, PI*h*h/3*(3*s.r - h));
}

// c is tuple containing containing the plane the circle lies
// on (pointing away from s1), the center of the circle,
// and the radius
int sphereSphereIntersection(Sphere3D s1, Sphere3D s2,
                          tuple<Plane3D, pt3, T> &c) {
  pt3 d = s2.o - s1.o; T d2 = norm(d);
  if (eq(d2, 0)) return eq(s1.r, s2.r) ? 2 : 0;
  T pd = (d2 + s1.r * s1.r - s2.r * s2.r) / 2;
  T h2 = s1.r * s1.r - pd * pd / d2; if (lt(h2, 0)) return 0;
  pt3 o = s1.o + d * pd / d2;
  c = make_tuple(Plane3D(d,o),o,sqrt(max(h2,T(0)))); return 1;
}

pair<T, T> sphereSphereIntersectionSAV(
    Sphere3D s1, Sphere3D s2) {
  pt3 d = s2.o - s1.o; T d2 = norm(d), dr = abs(s1.r - s2.r);
  T PI = acos(T(-1)); if (!lt(dr * dr, d2)) {
    T r = min(s1.r, s2.r);
    return make_pair(PI * 4 * r * r, PI * 4 * r * r * r / 3);
  }
  T sr = s1.r + s2.r;
  if (lt(sr * sr, d2)) return make_pair(T(0), T(0));
  T pd = (d2 + s1.r * s1.r - s2.r * s2.r) / 2;
  Plane3D pi = Plane3D(d, s1.o + d * pd / d2);
  pair<T, T> a = sphereHalfSpaceIntersectionSAV(s1, pi);
  pair<T, T> b = sphereHalfSpaceIntersectionSAV(
    s2, Plane3D(-pi.n, -pi.d));
  a.first += b.first; a.second += b.second; return a;
}
```

Polyhedron3D.h

Description: Polyhedron functions

"Point3D.h", "Sphere3D.h" 281b70, 59 lines

```
pt3 vectorArea2(const vector<pt3> &face) {
  pt3 ret(0, 0, 0); for (int i = 0, n = face.size(); i < n; i
      ++)
    ret += face[i] * face[(i + 1) % n];
  return ret;
}

T faceArea(const vector<pt3> &face) {
  return abs(vectorArea2(face)) / T(2);
}

// Flips some of the faces of a polyhedron such that they
// are oriented consistently
void reorient(vector<vector<pt3>> &faces) {
  int n = faces.size(); vector<vector<pair<int, bool>>> G(n);
  map<pair<pt3, pt3>, int> seen; for (int v = 0; v < n; v++)
    for (int i = 0, m = faces[v].size(); i < m; i++) {
      pt3 a = faces[v][i], b = faces[v][(i + 1) % m];
      auto it = seen.find(make_pair(a, b)); bool f = true;
      if (it == seen.end()) {
        it = seen.find(make_pair(b, a)); f = false;
      }
      if (it == seen.end()) seen[make_pair(a, b)] = v;
      else {
        int w = it->second; G[v].emplace_back(w, f);
        G[w].emplace_back(v, f);
      }
    }
  vector<char> flip(n, -1); vi q(n); int front = 0, back = 0;
  flip[q[back++] = 0] = 0; while (front < back) {
    int v = q[front++];
    for (auto &e : G[v]) if (flip[e.first] == -1)
      flip[q[back++] = e.first] = flip[v] ^ e.second;
  }
  for (int v = 0; v < n; v++) if (flip[v])
    reverse(faces[v].begin(), faces[v].end());
}

// faces must be the same orientation
```

```
T getSurfaceArea(const vector<vector<pt3>> &faces) {
  T sa = 0; for (auto &&face : faces) sa += faceArea(face);
  return sa;
}

// faces must be the same orientation
T getVolume6(const vector<vector<pt3>> &faces) {
  T vol6 = 0; for (auto &&face : faces)
    vol6 += (vectorArea2(face) | face[0]);
  return vol6;
}

int isInPolyhedron(const vector<vector<pt3>> &faces, pt3 p) {
  T sum = 0, PI = acos(T(-1)); Sphere3D s(p, 1);
  for (auto &&face : faces) {
    pt3 a = face[0], b = face[1], c = face[2], n = (b-a)*(c-a);
    if (eq((n | p) - (n | a), 0)) return 0;
    sum += remainder(s.surfaceAreaOnSph(face), 4 * PI);
  }
  return eq(sum, 0) ? -1 : 1;
}
```

ConvexHull3D.h

Description: Computes the convex hull of a set of N points 3D points (convex set of minimum points with the minimum volume)

"Point3D.h" 9b4fee, 66 lines

```
vector<vector<pt3>> convexHull3D(vector<pt3> P) {
  vector<array<int, 3>> hullInd;
  mt19937_64 rng(0); shuffle(P.begin(), P.end(), rng);
  int n = P.size(); for (int i = 1, num = 1; i < n; i++) {
    if (num == 1) {
      if (P[0] != P[i]) { swap(P[1], P[i]); num++; }
    } else if (num == 2) {
      if (!eq(norm((P[1] - P[0]) * (P[i] - P[0])), 0)) {
        swap(P[2], P[i]); num++;
      }
    } else if (num == 3) {
      if (above(P[0], P[1], P[2], P[i]) != 0) {
        swap(P[3], P[i]); num++;
      }
    }
  }
  vector<bool> active; vector<vector<int>> vis(n), rvis;
  vector<array<pair<int, int>, 3>> other; vi label(n, -1);
  auto addFace = [&] (int a, int b, int c) {
    hullInd.push_back({a, b, c}); active.push_back(true);
    rvis.emplace_back(); other.emplace_back();
  };
  auto addEdge = [&] (int a, int b) {
    vis[b].push_back(a); rvis[a].push_back(b);
  };
  auto abv = [&] (int a, int b) {
    array<int, 3> f = hullInd[a];
    return above(P[f[0]], P[f[1]], P[f[2]], P[b]) > 0;
  };
  auto edge = [&] (int f, int s) {
    return make_pair(hullInd[f][s], hullInd[f][(s + 1) % 3]);
  };
  auto glue = [&] (int af, int as, int bf, int bs) {
    other[af][as] = make_pair(bf, bs);
    other[bf][bs] = make_pair(af, as);
  };
  addFace(0, 1, 2); addFace(0, 2, 1);
  if (abv(1, 3)) swap(P[1], P[2]);
  for (int i = 0; i < 3; i++) glue(0, i, 1, 2 - i);
  for (int i = 3; i < n; i++) addEdge(abv(1, i), i);
  for (int i = 3; i < n; i++) {
    vector<int> rem; for (auto &&t : vis[i])
      if (active[t]) { active[t] = false; rem.push_back(t); }
    if (rem.empty()) continue;
    int st = -1; for (auto &&r : rem) rep(j, 0, 3) {
      int o = other[r][j].first; if (active[o]) {
        int a, b; tie(a, b) = edge(r, j); addFace(a,b,i); st=a;
        int cur = int(rvis.size()) - 1; label[a] = cur; vi tmp;
        set_union(rvis[r].begin(), rvis[r].end(),
                  rvis[o].begin(), rvis[o].end(),
                  back_inserter(tmp));
        for (auto &&x : tmp) if (abv(cur, x)) addEdge(cur, x);
        glue(cur, 0, other[r][j].first, other[r][j].second);
      }
    }
    for (int x = st, y; ; x = y) {
      int lx = label[x]; glue(lx,1,label[y=hullInd[lx][1]],2);
      if (y == st) break;
    }
  }
  vector<vector<pt3>> hull;
  for (int i = 0; i < int(hullInd.size()); i++) if (active[i])
    hull.push_back(vector<pt3>{P[hullInd[i][0]],
        P[hullInd[i][1]], P[hullInd[i][2]]});
  return hull;
}
```

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: $\mathcal{O}(n)$

d4375c, 16 lines

```
vi pi(const string& s) {
  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]);
  }
  return p;
}

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

Zfunc.h

Description: z[x] 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) {
  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]])
      z[i]++;
    if (i + z[i] > r)
      l = i, r = i + z[i];
  }
  return z;
}
```

Manacher.h

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

e7ad79, 13 lines

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

MinRotation.h

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

d07a42, 8 lines

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

SuffixArray.h

Description: Builds suffix array for a string. rnk is a vector of the ranks of the suffixes. ind is a vector of the indices in the original array of the suffixes sorted in lexicographical order. LCP is a vector of the longest common prefixes between the suffixes when sorted in lexicographical order.
Time: $\mathcal{O}(n \log n)$

66fdae, 35 lines

```
template <class _T> struct SuffixArray {
  using T = _T; int N; vector<int> ind, rnk, LCP;
  SuffixArray(const vector<T> &S)
      : N(S.size()), ind(N + 1), rnk(N + 1), LCP(N + 1) {
    vector<int> &tmp = LCP; iota(ind.begin(), ind.end(), 0);
    sort(ind.begin(), ind.begin() + N, [&] (int a, int b) {
      return S[a] < S[b];
    });
    rnk[ind[N]] = -1; for (int i = 0; i < N; i++) {
      rnk[ind[i]] = i > 0 && S[ind[i]] == S[ind[i - 1]]
          ? rnk[ind[i - 1]] : i;
    }
    for (int h = 1; h < N; h += h)
      for (int l = 0, r = 1; r <= N; r++) {
        if (rnk[ind[r - 1]] != rnk[ind[r]] && l + 1 < r) {
          sort(ind.begin() + l, ind.begin() + r,
               [&] (int a, int b) {
            return rnk[h + a] < rnk[h + b];
          });
          tmp[l] = l; for (int j = l + 1; j < r; j++)
            tmp[j] = rnk[h + ind[j - 1]] < rnk[h + ind[j]]
                ? j : tmp[j - 1];
          for (l++; l < r; l++) rnk[ind[l]] = tmp[l];
        } else if (rnk[ind[r - 1]] != rnk[ind[r]]) l++;
      }
    ind.pop_back(); rnk.pop_back(); tmp.pop_back();
    for (int i = 0, k = 0; i < N; i++) {
      if (rnk[i] == N - 1) { LCP[rnk[i]] = k = 0; continue; }
      int j = ind[rnk[i] + 1];
      while (i + k < N && j + k < N && S[i + k] == S[j + k])
        k++;
      if ((LCP[rnk[i]] = k) > 0) k--;
    }
  }
};
```

SuffixAutomaton.h

Description: Suffix Automaton with the root node at 0 Each distinct path from the root is a substring of the set of words inserted

a9c63e, 34 lines

```
template <class _T> struct SAMMapNode {
  using T = _T; int len, link; map<T, int> to;
  SAMMapNode(int len) : len(len), link(-1) {}
  bool hasEdge(const T &a) const { return to.count(a); }
  int getEdge(const T &a) const { return to.at(a); }
  void setEdge(const T &a, int n) { to[a] = n; }
};

template <class Node> struct SuffixAutomaton {
  using T = typename Node::T; vector<Node> TR; int last;
  SuffixAutomaton() : TR(1, Node(0)), last(0) {}
  void add(const T &a) {
    int u = -1; if (!TR[last].hasEdge(a)) {
      u = TR.size(); TR.emplace_back(TR[last].len + 1);
      for (; last != -1 && !TR[last].hasEdge(a);
            last = TR[last].link)
        TR[last].setEdge(a, u);
      if (last == -1) { TR[last = u].link = 0; return; }
    }
    int p = TR[last].getEdge(a);
    if (TR[p].len == TR[last].len + 1) {
      (u == -1 ? last : TR[last = u].link) = p; return;
    }
    int q = TR.size(); TR.push_back(TR[p]);
    TR[q].len = TR[last].len + 1;
    TR[p].link = q; if (u != -1) TR[u].link = q;
    while (last != -1 && TR[last].hasEdge(a)
          && TR[last].getEdge(a) == p) {
      TR[last].setEdge(a, q); last = TR[last].link;
    }
    last = u == -1 ? q : u;
  }
  void terminate() { last = 0; }
};
```

PalTree.h

Description: Palindromic Tree with two roots at 0 (with length -1) and 1 (with length 0)

fe8ff6, 49 lines

```
template <class _T> struct PalTreeMapNode {
  using T = _T; int len, link, qlink; map<T, int> to;
  PalTreeMapNode(int len) : len(len), link(1), qlink(1) {}
  int getEdge(const T &a) const {
    auto it = to.find(a);
    return it == to.end() ? 1 : it->second;
  }
  void setEdge(const T &a, int n) { to[a] = n; }
};

template <class Node> struct PalindromicTree {
  using T = typename Node::T;
  T def; vector<T> S; vector<Node> TR;
  vector<int> last, modified;
  PalindromicTree(const T &def)
      : def(def), S(1, def),
        TR(vector<Node>{Node(-1), Node(0)}), last(1, 1) {
    TR[1].link = TR[1].qlink = 0;
  }
  int getLink(int x, int i) {
    while (S[i - 1 - TR[x].len] != S[i])
      x = S[i - 1 - TR[TR[x].link].len] == S[i]
          ? TR[x].link : TR[x].qlink;
    return x;
  }
  void add(const T &a) {
    int i = S.size(); S.push_back(a);
    int p = getLink(last.back(), i);
    modified.push_back(-1); if (TR[p].getEdge(a) == 1) {
      int u = TR.size(); TR.emplace_back(TR[p].len + 2);
      TR[u].link = TR[getLink(TR[p].link, i)].getEdge(a);
      T b = S[i - TR[TR[u].link].len];
      T c = S[i - TR[TR[TR[u].link].link].len];
      TR[u].qlink = b == c ? TR[TR[u].link].qlink : TR[u].link;
      TR[modified.back() = p].setEdge(a, u);
    }
    last.push_back(TR[p].getEdge(a));
  }
  void terminate() {
    S.push_back(def); last.push_back(1);
    modified.push_back(-1);
  }
  void undo() {
    if (modified.back() != -1) {
      TR[modified.back()].setEdge(S.back(), 1); TR.pop_back();
    }
    S.pop_back(); last.pop_back(); modified.pop_back();
  }
};
```

Various (10)

10.1 Intervals

IntervalContainer.h

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

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

edce47, 23 lines

```
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
  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);
  }
  if (it != is.begin() && (--it)->second >= L) {
    L = min(L, it->first);
    R = max(R, it->second);
    is.erase(it);
  }
  return is.insert(before, {L,R});
}

void removeInterval(set<pii>& is, int L, int R) {
  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;
  if (R != r2) is.emplace(R, r2);
}
```

ConstantIntervals.h

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

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

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

753a4c, 19 lines

```
template<class F, class G, class T>
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;
  } else {
    int mid = (from + to) >> 1;
    rec(from, mid, f, g, i, p, f(mid));
    rec(mid+1, to, f, g, i, p, q);
  }
}
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);
  rec(from, to-1, f, g, i, p, q);
  g(i, to, q);
}
```

10.2 Misc. algorithms

TernarySearch.h

Description: Ternary search for the maximum of a function

b03dcd, 9 lines

```
template <class T, class F, class Cmp = less<T>>
T tsearch(T lo, T hi, F f, int iters, Cmp cmp = less<T>()) {
  for (int it = 0; it < iters; it++) {
    T m1 = lo + (hi - lo) / 3, m2 = hi - (hi - lo) / 3;
    if (cmp(f(m1), f(m2))) lo = m1;
    else hi = m2;
  }
  return lo + (hi - lo) / 2;
}
```

LIS.h

Description: Compute indices for the longest increasing subsequence.

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

2932a0, 17 lines

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

FastKnapsack.h

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

Time: $\mathcal{O}\left(N \max(w_i)\right)$ b20ccc, 16 lines

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

10.3 Dynamic programming

KnuthDP.h

Description: Must satisfy dp[l][r] = max(dp[l][m] + dp[m][r] + cost(l, m, r) for l <= m <= r and max(l + 1, opt[l][r - 1]) <= opt[l][r] <= min(opt[l + 1][r], r - 1), where opt[l][r] is the optimal value of m for dp[l][r]
Time: $\mathcal{O}\left(N^2\right)$ d5d243, 18 lines

```
template <class T, class F, class Cmp = less<T>>
vector<vector<T>> knuth(int N, F f, Cmp cmp = Cmp()) {
  vector<vector<T>> dp(N, vector<T>(N + 1, T()));
  vector<vector<int>> opt(N, vector<int>(N + 1));
  for (int l = N - 1; l >= 0; l--)
    for (int r = l; r <= N; r++) {
      if (r - l <= 1) { opt[l][r] = l; continue; }
      int st = max(l + 1, opt[l][r - 1]);
      int en = min(opt[l + 1][r], r - 1);
      for (int m = st; m <= en; m++) {
        T cost = dp[l][m] + dp[m][r] + f(l, m, r);
        if (m == st || cmp(dp[l][r], cost)) {
          dp[l][r] = cost; opt[l][r] = m;
        }
      }
    }
  return dp;
}
```

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \le k < hi(i)}(f(i,k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.
Time: $\mathcal{O}\left((N + (hi - lo)) \log N\right)$ d38d2b, 18 lines

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

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

10.4 Debugging tricks

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

10.5 Optimization tricks

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

10.5.1 Bit hacks

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

FastInput.h

Description: Read an integer from stdin. Usage requires your program to pipe in input from file.
Usage: `./a.out < input.txt`
Time: About 5x as fast as cin/scanf. 7b3c70, 17 lines

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

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

BumpAllocator.h

Description: When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation. 745db2, 8 lines

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

SmallPtr.h

Description: A 32-bit pointer that points into BumpAllocator memory.
"BumpAllocator.h" 2dd6c9, 10 lines

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

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers.
Usage: `vector<vector<int, small<int>>> ed(N);` bb66d4, 14 lines

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

template<class T> struct small {
  typedef T value_type;
  small() {}
  template<class U> small(const U&) {}
  T* allocate(size_t n) {
    buf_ind -= n * sizeof(T);
    buf_ind &= 0 - alignof(T);
    return (T*)(buf + buf_ind);
  }
  void deallocate(T*, size_t) {}
};
```