Massachusetts Institute of Technology

MIT Taxi

Jacob Teo, Siyong Huang, Thomas Guo

2024-04-12

Contest (1)

template.cpp
16 lines
```cpp
#include <bits/stdc++.h>
using namespace std;

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

bool ckmax(auto &a, auto const& b) {return b>a?a=b,1:0;}
bool ckmin(auto &a, auto const& b) {return b<a?a=b,1:0;}

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

.bashrc
5 lines
```bash
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
    -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps =
    <>
setxkbmap -option caps:escape # bind caps to esc
setxkbmap -option # remove options
```

.vimrc
17 lines
```vim
set ts=2 sw=2 ai cin nu rnu udf udir=~/.vim/udir
sy on
set cul ru nowrap wmnu sc is bs=indent,eol,start cino=g0
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
    \| md5sum \| cut -c-6

nmap <F8> :w <Bar> !g++ -std=c++20 -DLOCAL %<CR>
nmap <F9> :w <Bar> !g++ -std=c++20 -DLOCAL % && ./a.out<CR>

autocmd FileType python set sw=4 ts=4 sts=4 et nocin si
autocmd FileType python nmap <F8> :w <Bar> !python3 "%"<CR>
autocmd FileType python nmap <F9> :w <Bar> !python3 -i "%"<
    CR>

" To map caps lock -> escape --
" setxkbmap -option caps:escape
```

hash.sh
3 lines
```bash
# Hashes a file, ignoring all whitespace and comments. Use
    for
# verifying that code was correctly typed.
cpp-11 -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |
    cut -c-6
```

troubleshoot.txt
52 lines
```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it
    .

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

Mathematics (2)

2.1 Equations

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

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

where A'_i is A with the i'th column replaced by b.

2.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, 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. $a_n = (d_1 n + d_2) r^n$.

2.3 Trigonometry

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

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

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

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

2.4 Geometry

2.4.1 Triangles

For side lengths a, b, c, and $p = \frac{a+b+c}{2}$,

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

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

2.4.2 Quadrilaterals

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

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

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

2.4.3 Pick's Theorem

Polygon with integer vertices: $A = i + \frac{b}{2} - 1$

2.5 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1 - x^2}} \qquad \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1 - x^2}}$$

$$\frac{d}{dx} \tan x = 1 + \tan^2 x \qquad \frac{d}{dx} \arctan x = \frac{1}{1 + x^2}$$

$$\int \tan ax = -\frac{\ln |\cos ax|}{a} \qquad \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \text{erf}(x) \qquad \int x e^{ax} dx = \frac{e^{ax}}{a^2}(ax - 1)$$

2.6 Sums

$$\sum_{i=1}^{n} i^3 = \frac{n^2(n + 1)^2}{4} ; \sum_{i=1}^{n} i^4 = \frac{n(n + 1)(2n + 1)(3n^2 + 3n - 1)}{30}$$

2.7 Series

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

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

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

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

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

$$\frac{1}{1 - x} = 1 + x + x^2 + \ldots = \sum_{n \geq 0} x^n$$

$$-\ln(1 - x) = x + \frac{x^2}{2} + \frac{x^3}{3} + \ldots = \sum_{n \geq 1} \frac{x^n}{n}$$

$$(1 + x)^r = \sum_{n \geq 0} \binom{r}{n} x^n$$

2.8 Probability theory

2.8.1 Discrete distributions

Binomial distribution

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

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

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

First success distribution

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

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

Poisson distribution

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

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

2.8.2 Continuous distributions

Exponential distribution

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

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

Normal distribution

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

2.9 Markov chains

Transition matrix: $\mathbf{P} = (p_{ij})$, with $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$ π is a stationary distribution if $\pi = \pi\mathbf{P}$. If *irreducible* (any state to any state possible): $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i. π_j/π_i is the expected number of visits in state j between two visits in state i. For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i's degree. A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k\to\infty} \mathbf{P}^k = \mathbf{1}\pi$. A Markov chain is an A-chain if the states can be partitioned into two sets $\mathbf{A}$ and $\mathbf{G}$, such that all states in $\mathbf{A}$ are absorbing ($p_{ii} = 1$), and all states in $\mathbf{G}$ leads to an absorbing state in $\mathbf{A}$. The probability for absorption in state $i \in \mathbf{A}$, when the initial state is j, is
$a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik} p_{kj}$. The expected time until absorption, when the initial state is i, is $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$.

2.10 Graphs

2.10.1 Erdos-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 $k = 1 \ldots n$,

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

2.10.2 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).

Data structures (3)

FastStaticRMQ.h
Description: Static RMQ min(V[a], V[a + 1], ... V[b]) in constant time.
Usage: RMQ rmq(values); rmq.query(inclusive, exclusive);
Time: $\mathcal{O}(N + Q)$ fda446, 28 lines

```cpp
template<typename T> struct RMQ {
  vector<T> v;
  int n; static const int b = 30;
  vector<int> mask, t;

  int op(int x, int y) { return v[x] < v[y] ? x : y; }
  int msb(int x) { return __builtin_clz(1)-__builtin_clz(x); }

  int small(int r, int sz = b) { return r-msb(mask[r]&((1<<
    sz)-1)); }
  rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(
    n) {
    for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
      at = (at<<1)&((1<<b)-1);
      while (at and op(i, i-msb(at&-at)) == i) at ^= at&-at;
    }
    for (int i = 0; i < n/b; i++) t[i] = small(b*i+b-1);
```

```cpp
    for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0; i+(1
      <<j) <= n/b; i++)
      t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j-1)+i+(1<<(j-1
        ))]);
  }
  T query(int l, int r) {
    if (r-l+1 <= b) return v[small(r, r-l+1)];
    int ans = op(small(l+b-1), small(r));
    int x = l/b+1, y = r/b-1;
    if (x <= y) {
      int j = msb(y-x+1);
      ans = op(ans, op(t[n/b*j+x], t[n/b*j+y-(1<<j)+1]));
    }
    return v[ans];
  }
};
```

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

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

FenwickTree2d.h
Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).
Time: $\mathcal{O}(\log^2 N)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)
"FenwickTree.h"   157f07, 22 lines

```cpp
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;
  }
};
```

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

```cpp
#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});
```

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

```cpp
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;
  }
};
```

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

```cpp
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; }
};
```

```cpp
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;
  }
};
```

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

```cpp
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]);
```

```cpp
    if (end) res[qi] = calc();
  }
  return res;
}
```

PBBST.h

Description: Persistent AVL tree with split

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

852300, 82 lines

```cpp
struct PAVL {
  struct Node {
    int t;
    int s, h; /* Customize */
    array<int, 2> c;
    int val;
    Node() : t(), s(1), h(1), c{ -1, -1 } { t = T; }
    Node(int _val) : Node() { val = _val; }
    void up() {}; /* Customize */
    void down() {};
  };
  static vector<Node> N;
  static int T;
  static int clone(int n) {
    if (n == -1) return -1; //assert(N[n].t >= t);
    if (N[n].t == T) return n;
    return N.push_back(N[n]), N.back().t = T, N.size() - 1;
  }
  static int gh(int n) { return n != -1 ? N[n].h : 0; }
  static int gs(int n) { return n != -1 ? N[n].s : 0; }
  static void up(int n) {
    N[n].h = max(gh(N[n].c[0]), gh(N[n].c[1])) + 1;
    N[n].s = gs(N[n].c[0]) + gs(N[n].c[1]) + 1;
    N[n].up();
  }
  static int down(int n) { n = clone(n); return N[n].down(),
    n; }
  static int rotate(int n, int d) {
    n = clone(n); int o = down(N[n].c[d]);
    N[n].c[d] = N[o].c[!d], N[o].c[!d] = n;
    up(n), up(o);
    return o;
  }
  static int balance(int n) {
    assert(N[n].t == T); up(n);
    int diff = gh(N[n].c[0]) - gh(N[n].c[1]), d;
    if (diff >= 2) d = 0;
    else if (diff <= -2) d = 1;
    else return n;
    N[n].c[d] = down(N[n].c[d]);
    if (gh(N[N[n].c[d]].c[d]) + 1 < gh(N[n].c[d]))
      N[n].c[d] = rotate(N[n].c[d], !d);
    return rotate(n, d);
  }
  static int merge_root(int l, int n, int r) {
    if (gh(l) + 1 < gh(r))
      return r = down(r), N[r].c[0] = merge_root(l, n, N[r].
        c[0]), balance(r);
    else if (gh(r) + 1 < gh(l))
      return l = down(l), N[l].c[1] = merge_root(N[l].c[1],
        n, r), balance(l);
    else return N[n].c = { l, r }, balance(n);
  }
  static tuple<int, int> split(int n, int k) {
    if (n != -1) n = down(n);
    if (k == 0) return { -1, n };
    if (k == N[n].s) return { n, -1 };
    if (k <= gs(N[n].c[0])) {
      auto [l, r] = split(N[n].c[0], k);
      return { l, merge_root(r, n, N[n].c[1]) };
    } else {
      auto [l, r] = split(N[n].c[1], k - gs(N[n].c[0]) - 1);
      return { merge_root(N[n].c[0], n, l), r };
    }
  }
  static int merge(int l, int r) {
    if (r == -1) return l;
    auto [x, nr] = split(r, 1);
    return merge_root(l, clone(x), nr);
  }
  PAVL(int v) : root(v) {}
  int root;
  PAVL() : root(-1) {}
  PAVL(Node&& n) : root(N.size()) { N.push_back(n); }
```

```
72   friend PAVL operator+(PAVL a, PAVL b) { return merge(a.
        root, b.root); }
73   tuple<PAVL, PAVL> split(int k) {
74      auto [l, r] = split(root, k);
75      return { PAVL(l), PAVL(r) };
76   }
77   PAVL step() { ++T; return clone(root); }
78   Node& get_root() { return N[root]; }
79 };
80 typedef PAVL::Node Node;
81 vector<Node> PAVL::N;
82 int PAVL::T;
```

PBDS.h
Description: examples for PBDS BBST, mergeable heaps and rope.
Time: $\mathcal{O}(\log N)$
<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp>,
<ext/pb_ds/priority_queue.hpp>, <ext/rope>   35b953, 38 lines

```
1 using namespace std;
2 using namespace __gnu_pbds;
3 using namespace __gnu_cxx;
4 template<class T>
5 using Tree = tree<T, null_type, less<T>, rb_tree_tag,
        tree_order_statistics_node_update>;
6 template<class T>
7 using Heap = __gnu_pbds::priority_queue<T, less<T>,
        pairing_heap_tag>;
8 //binary_heap_tag, pairing_heap_tag, binomial_heap_tag,
        rc_binomial_heap_tag, thin_heap_tag
9 void pbds() {
10   Tree<int> t, t2; t.insert(8);
11   auto it = t.insert(10).first;
12   assert(it == t.lower_bound(9));
13   assert(t.order_of_key(10) == 1);
14   assert(t.order_of_key(11) == 2);
15   assert(*t.find_by_order(0) == 8);
16   t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
17
18   Heap<int> pq1, pq2;
19   pq1.push(1); pq2.push(5);
20   pq1.join(pq2); // merge pq2 into pq1
21   assert(pq1.top()==5);
22   auto pq_it = pq1.push(3);
23   assert(pq1.top()==5);
24   pq1.modify(pq_it,7); // modify-key in O(log N)
25   assert(pq1.top()==7);
26
27   int n=3;
28     rope<int> v(n, 0);
29     for (int i=0; i<n; i++) v.mutable_reference_at(i) = i +
        1; // (1 2 3)
30   for (int i=0; i<n; i++) v.push_back(i + n + 1); // (1 2 3
        4 5 6)
31     int l=1, r=3;
32     rope<int> cur = v.substr(l, r-l+1); // 2 3 4
33     v.erase(l, r-l+1); // 1 5 6
34   v.insert(v.mutable_begin() + 2, cur);
35     v.insert(v.mutable_begin(), cur); //to start (2 3 4 1 5
        6)
36     // v.insert(v.mutable_reference_at(0), cur); // to ONE
        AFTER start (1 2 3 4 5 6)
37     // v.insert(v.mutable_begin() + 2, cur); // to TWO AFTER
        start (1 5 2 3 4 6)
38 }
```

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

```
1 template<class T>
2 struct RMQ {
3   vector<vector<T>> jmp;
4   RMQ(const vector<T>& V) : jmp(1, V) {
5     for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
6       jmp.emplace_back(sz(V) - pw * 2 + 1);
7       rep(j,0,sz(jmp[k]))
8         jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
9     }
10   }
```

```
11   T query(int a, int b) {
12     assert(a < b); // or return inf if a == b
13     int dep = 31 - __builtin_clz(b - a);
14     return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
15   }
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

```
1 struct Tree {
2   typedef int T;
3   static constexpr T unit = INT_MIN;
4   T f(T a, T b) { return max(a, b); } // (any associative fn
        )
5   vector<T> s; int n;
6   Tree(int n = 0, T def = unit) : s(2*n, def), n(n) {}
7   void update(int pos, T val) {
8     for (s[pos += n] = val; pos /= 2;)
9       s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
10   }
11   T query(int b, int e) { // query [b, e)
12     T ra = unit, rb = unit;
13     for (b += n, e += n; b < e; b /= 2, e /= 2) {
14       if (b % 2) ra = f(ra, s[b++]);
15       if (e % 2) rb = f(s[--e], rb);
16     }
17     return f(ra, rb);
18   }
19 };
```

SparseSegTree2D.h
Description: 2D point-update range-query segtree supporting 10^9 coordinates (IOI 2013 game)
Time: $\mathcal{O}(\log^2 N)$ 40b942, 92 lines

```
1 #define DEFAULT 0ll
2 ll func(ll a, ll b) {return max(a, b);} // associative func
3 struct SegTree2D {
4   int R,C,root;
5   vector<int> l,r,b,e,st;
6   vector<ll> v;
7   inline int mid(int x,int y) {return ((x+y)>>1);}
8   SegTree2D(int _R,int _C):R(_R),C(_C) {
9     l.pb(0),r.pb(0),b.pb(0),e.pb(0),st.pb(0),v.pb(DEFAULT);
        R=_R;
10     root=alloc2(0,R-1);
11   }
12   int alloc(int _b,int _e,ll _v) {
13     l.pb(0),r.pb(0),b.pb(_b),e.pb(_e),st.pb(0),v.pb(_v);
14     return sz(b)-1;
15   }
16   void lca(int b,int e,int ob,int oe,int i,int &nb,int &ne)
        {
17     int m=mid(b,e);
18     if ((i<=m&&ob>m)||(i>m&&oe<=m)) nb=b,ne=e;
19     else (i>m)?lca(m+1,e,ob,oe,i,nb,ne):lca(b,m,ob,oe,i,nb,
        ne);
20   }
21   void up(int x) {v[x]=func(v[l[x]],v[r[x]]);}
22   void update1(int x,int i,ll nv) {
23     if (b[x]>i||e[x]<i) return;
24     if (b[x]==e[x]) {
25       v[x]=nv;
26       return;
27     }
28     int m=mid(b[x],e[x]);
29     if (i<=m) {
30       if (l[x]) {
31         if (b[l[x]]<=i && i<=e[l[x]]) update1(l[x],i,nv);
32         else {
33           int nb,ne;
34           lca(0,C-1,b[l[x]],e[l[x]],i,nb,ne);
35           int y=l[x];
36           l[x]=alloc(nb,ne,DEFAULT);
37           if (i>mid(nb,ne)) l[l[x]]=y,r[l[x]]=alloc(i,i,nv);
38           else r[l[x]]=y,l[l[x]]=alloc(i,i,nv);
39           up(l[x]);
40         }
41       } else l[x]=alloc(i,i,nv);
42     } else {
43       if (r[x]) {
44         if (b[r[x]]<=i && i<=e[r[x]]) update1(r[x],i,nv);
45         else {
46           int nb,ne;
47           lca(0,C-1,b[r[x]],e[r[x]],i,nb,ne);
48           int y=r[x];
49           r[x]=alloc(nb,ne,DEFAULT);
50           if (i>mid(nb,ne)) l[r[x]]=y,r[r[x]]=alloc(i,i,nv);
51           else r[r[x]]=y,l[r[x]]=alloc(i,i,nv);
52           up(r[x]);
53         }
54       } else r[x]=alloc(i,i,nv);
55     }
56     up(x);
57   }
58   ll query1(int x,int qb,int qe) {
59     if (!x) return DEFAULT;
60     if (b[x]>qe||e[x]<qb) return DEFAULT;
61     if (b[x]>=qb&&e[x]<=qe) return v[x];
62     return func(query1(l[x],qb,qe),query1(r[x],qb,qe));
63   }
64   int alloc2(int _b,int _e) {
65     int newnode = alloc(0,C-1,DEFAULT);
66     l.pb(0),r.pb(0),b.pb(_b),e.pb(_e),v.pb(DEFAULT),st.pb(
        newnode);
67     return sz(b)-1;
68   }
69   void update2(int x,int i,int j,ll nv) {
70     if (b[x]>i||e[x]<i) return;
71     if (b[x]==e[x]) update1(st[x],j,nv);
72     else {
73       int m=mid(b[x],e[x]);
74       if (!l[x]) {
75         l[x]=alloc2(b[x],m);
76         r[x]=alloc2(m+1,e[x]);
77       }
78       if (i<=m) update2(l[x],i,j,nv);
79       else update2(r[x],i,j,nv);
80       update1(st[x],j,func(query1(st[l[x]],j,j),query1(st[r[
        x]],j,j)));
81     }
82   }
83   ll query2(int x,int rb,int re,int cb,int ce) {
84     if (!x) return DEFAULT;
85     if (b[x]>re||e[x]<rb) return DEFAULT;
86     if (b[x]>=rb&&e[x]<=re) return query1(st[x],cb,ce);
87     return func(query2(l[x],rb,re,cb,ce),query2(r[x],rb,re,
        cb,ce));
88   }
89   void update(int p,int q,ll k) {update2(root,p,q,k);}
90   ll query(int p,int q,int u,int v) {return query2(root,p,u,
        q,v);}
91 };
```

Treap-benq.h
Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data. 0-index.
Time: $\mathcal{O}(\log N)$ c8a465, 72 lines

```
1 using pt = struct Node*;
2 struct Node {
3   int pri, val; pt c[2]; // essential
4   int sz; ll sum; // for range queries
5   bool flip = 0; // lazy update
6   Node(int _val) {
7     pri = rand(); sum = val = _val;
8     sz = 1; c[0] = c[1] = nullptr;
9   }
10   ~Node() { rep(i,0,2) delete c[i]; }
11 };
12 int getsz(pt x) { return x?x->sz:0; }
13 ll getsum(pt x) { return x?x->sum:0; }
14 pt prop(pt x) { // lazy propagation
15   if (!x || !x->flip) return x;
16   swap(x->c[0],x->c[1]);
17   x->flip = 0; rep(i,0,2) if (x->c[i]) x->c[i]->flip ^= 1;
18   return x;
19 }
20 pt calc(pt x) {
21   pt a = x->c[0], b = x->c[1];
22   assert(!x->flip); prop(a), prop(b);
```

```cpp
23    x->sz = 1+getsz(a)+getsz(b);
24    x->sum() = x->val+getsum(a)+getsum(b);
25    return x;
26  }
27  void tour(pt x, vi& v) { // print values of nodes,
28    if (!x) return; // inorder traversal
29    prop(x); tour(x->c[0],v); v.pb(x->val); tour(x->c[1],v);
30  }
31  pair<pt,pt> split(pt t, int v) { // >= v goes to the right
32    if (!t) return {t,t};
33    prop(t);
34    if (t->val >= v) {
35      auto p = split(t->c[0], v); t->c[0] = p.second;
36      return {p.first,calc(t)};
37    } else {
38      auto p = split(t->c[1], v); t->c[1] = p.first;
39      return {calc(t),p.second};
40    }
41  }
42  pair<pt,pt> splitsz(pt t, int sz) { // sz nodes go to left
43    if (!t) return {t,t};
44    prop(t);
45    if (getsz(t->c[0]) >= sz) {
46      auto p = splitsz(t->c[0],sz); t->c[0] = p.second;
47      return {p.first,calc(t)};
48    } else {
49      auto p=splitsz(t->c[1],sz-getsz(t->c[0])-1); t->c[1]=p.
           first;
50      return {calc(t),p.second};
51    }
52  }
53  pt merge(pt l, pt r) { //  keys in l < keys in r
54    if (!l || !r) return l?:r;
55    prop(l), prop(r); pt t;
56    if (l->pri > r->pri) l->c[1] = merge(l->c[1],r), t = l;
57    else r->c[0] = merge(l,r->c[0]), t = r;
58    return calc(t);
59  }
60  pt ins(pt x, int v) { // insert v
61    auto a = split(x,v), b = split(a.second,v+1);
62    return merge(a.first,merge(new Node(v),b.second)); }
63  pt del(pt x, int v) { // delete v
64    auto a = split(x,v), b = split(a.second,v+1);
65    return merge(a.first,b.second); }
66  pt inspos(pt t, pt n, int pos) {// insert so node is in pos
           (0-indexed)
67    auto pa = splitsz(t, pos);
68    return merge(merge(pa.first, n), pa.second); }
69  pt delpos(pt t, int pos) {
70    auto pa = splitsz(t, pos);
71    auto pb = splitsz(pa.second, 1);
72    return merge(pa.first, pb.second); }
```

UnionFind.h
Description: Disjoint-set data structure.
Time: $\mathcal{O}(\alpha(N))$
                                            7aa27c, 14 lines

```cpp
1  struct UF {
2    vi e;
3    UF(int n) : e(n, -1) {}
4    bool sameSet(int a, int b) { return find(a) == find(b); }
5    int size(int x) { return -e[find(x)]; }
6    int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]);
          }
7    bool join(int a, int b) {
8      a = find(a), b = find(b);
9      if (a == b) return false;
10     if (e[a] > e[b]) swap(a, b);
11     e[a] += e[b]; e[b] = a;
12     return true;
13   }
14 };
```

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

```cpp
1  struct RollbackUF {
2    vi e; vector<pii> st;
3    RollbackUF(int n) : e(n, -1) {}
4    int size(int x) { return -e[find(x)]; }
```

```cpp
5    int find(int x) { return e[x] < 0 ? x : find(e[x]); }
6    int time() { return sz(st); }
7    void rollback(int t) {
8      for (int i = time(); i --> t;)
9        e[st[i].first] = st[i].second;
10     st.resize(t);
11   }
12   bool join(int a, int b) {
13     a = find(a), b = find(b);
14     if (a == b) return false;
15     if (e[a] > e[b]) swap(a, b);
16     st.push_back({a, e[a]});
17     st.push_back({b, e[b]});
18     e[a] += e[b]; e[b] = a;
19     return true;
20   }
21 };
```

Numerical (4)

4.1 Polynomials and recurrences

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

```cpp
1  vector<double> polyRoots(Poly p, double xmin, double xmax) {
2    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
3    vector<double> ret;
4    Poly der = p;
5    der.diff();
6    auto dr = polyRoots(der, xmin, xmax);
7    dr.push_back(xmin-1);
8    dr.push_back(xmax+1);
9    sort(all(dr));
10   rep(i,0,sz(dr)-1) {
11     double l = dr[i], h = dr[i+1];
12     bool sign = p(l) > 0;
13     if (sign ^ (p(h) > 0)) {
14       rep(it,0,60) { // while (h - l > 1e-8)
15         double m = (l + h) / 2, f = p(m);
16         if ((f <= 0) ^ sign) l = m;
17         else h = m;
18       }
19       ret.push_back((l + h) / 2);
20     }
21   }
22   return ret;
23 }
```

PolyOps.h
Description: Operations on formal power series
                                         a8f86e, 50 lines

```cpp
1  int const RT = 5;
2  using T = mint;
3  using poly = vector<mint>;
4  void remz(poly& p) { while (sz(p)&&p.back().v==0) p.pop_back
          (); }
5  poly REMZ(poly p) { remz(p); return p; }
6  poly rev(poly p) { reverse(all(p)); return p; }
7  poly shift(poly p, int x) {
8    if (x >= 0) p.insert(begin(p),x,0);
9    else assert(sz(p)+x >= 0), p.erase(begin(p),begin(p)-x);
10   return p;
11 }
12 poly RSZ(const poly& p, int x) {
13   if (x <= sz(p)) return poly(begin(p),begin(p)+x);
14   poly q = p; q.resize(x); return q;   }
15 T eval(const poly& p, T x) { // evaluate at point x
16   T res = 0; for (int i = sz(p)-1; i>=0; i--) res = x*res+p[
          i];
17   return res; }
18 poly dif(const poly& p) { // differentiate
19   poly res; rep(i,1,sz(p)) res.pb(T(i)*p[i]);
20   return res; }
21 poly integ(const poly& p) { // integrate
22   static poly invs{0,1};
23   for (int i = sz(invs); i <= sz(p); ++i)
24     invs.pb(-MOD/i*invs[MOD%i]);
```

```cpp
25   poly res(sz(p)+1); rep(i,0,sz(p)) res[i+1] = p[i]*invs[i+1
          ];
26   return res;
27 }
28 poly& operator+=(poly& l, const poly& r) {
29   l.resize(max(sz(l),sz(r))); rep(i,0,sz(r)) l[i] += r[i];
30   return l; }
31 poly& operator-=(poly& l, const poly& r) {
32   l.resize(max(sz(l),sz(r))); rep(i,0,sz(r)) l[i] -= r[i];
33   return l; }
34 poly& operator*=(poly& l, const T& r) { for (auto &t:l) t *=
          r;
35   return l;  }
36 poly& operator/=(poly& l, const T& r) { for (auto &t:l) t /=
          r;
37   return l; }
38 poly operator+(poly l, const poly& r) { return l += r; }
39 poly operator-(poly l, const poly& r) { return l -= r; }
40 poly operator-(poly l) { for (auto &t:l) t *= -1; return l;
          }
41 poly operator*(poly l, const T& r) { return l *= r; }
42 poly operator*(const T& r, const poly& l) { return l*r; }
43 poly operator/(poly l, const T& r) { return l /= r; }
44 poly operator*(const poly& l, const poly& r) {
45   if (!min(sz(l),sz(r))) return {};
46   poly x(sz(l)+sz(r)-1);
47   rep(i,0,sz(l)) rep(j,0,sz(r)) x[i+j] += l[i]*r[j];
48   return x;
49 }
50 poly& operator*=(poly& l, const poly& r) { return l = l*r; }
```

PolyOps2.h
Description: Operations on formal power series
                                        9e805d, 87 lines

```cpp
1  void fft(vector<T>& A, bool inverse = 0) { // NTT
2    int n = sz(A); assert((MOD-1)%n == 0); vector<T> B(n);
3    for(int b = n/2; b; b /= 2, swap(A,B)) { // w = n/b'th
          root
4      T w = pow(mint(RT),(MOD-1)/n*b), m = 1;
5      for(int i = 0; i < n; i += b*2, m *= w) rep(j,0,b) {
6        T u = A[i+j], v = A[i+j+b]*m;
7        B[i/2+j] = u+v; B[i/2+j+n/2] = u-v;
8      }
9    }
10   if (inverse) { reverse(1+all(A));
11     T z = invert(T(n)); for (auto &t:A) t *= z; }
12 } // for NTT-able moduli -- 3397f7
13 vector<T> conv(vector<T> A, vector<T> B) {
14   if (!min(sz(A),sz(B))) return {};
15   int s = sz(A)+sz(B)-1, n = 1; for (; n < s; n *= 2);
16   A.resize(n), fft(A); B.resize(n), fft(B);
17   rep(i,0,n) A[i] *= B[i];
18   fft(A,1); A.resize(s); return A;
19 } // 7956e1
20 poly inv(poly A, int n) { // Q-(1/Q-A)/(-Q^{-2})
21   poly B{invert(A[0])};
22   for (int x = 2; x/2 < n; x *= 2)
23     B = 2*B-RSZ(conv(RSZ(A,x),conv(B,B)),x);
24   return RSZ(B,n);
25 } // 455ce0
26 poly sqrt(const poly& A, int n) {  // Q-(Q^2-A)/(2Q)
27   assert(A[0].v == 1); poly B{1};
28   for (int x = 2; x/2 < n; x *= 2)
29     B = invert(T(2))*RSZ(B+conv(RSZ(A,x),inv(B,x)),x);
30   return RSZ(B,n);
31 } // ecb486
32 // return {quotient, remainder}
33 pair<poly,poly> quoRem(const poly& f, const poly& g) {
34   if (sz(f) < sz(g)) return {{},f};
35   poly q = conv(inv(rev(g),sz(f)-sz(g)+1),rev(f));
36   q = rev(RSZ(q,sz(f)-sz(g)+1));
37   poly r = RSZ(f-conv(q,g),sz(g)-1); return {q,r};
38 } // bffc5b
39 poly log(poly A, int n) { assert(A[0].v == 1); // (ln A)' =
          A'/A
40   A.resize(n); return integ(RSZ(conv(dif(A),inv(A,n-1)),n-1)
          );
41 } // bda418
42 poly exp(poly A, int n) { assert(A[0].v == 0);
43   poly B{1}, IB{1}; // inverse of B
44   for (int x = 1; x < n; x *= 2) {
45     IB = 2*IB-RSZ(conv(B,conv(IB,IB)),x);
```

```
46    poly Q = dif(RSZ(A,x)); Q += RSZ(conv(IB,dif(B)-conv(B,Q
        )),2*x-1);
47    B = B+RSZ(conv(B,RSZ(A,2*x)-integ(Q)),2*x);
48  }
49  return RSZ(B,n);
50 } // 203953
51 poly pow(poly A, ll b, int n) {
52    if (b==0) { poly r(n,0); r[0]=1; return r; }
53    int t = -1;
54    for (int i = 0; i < n; i++) if (A[i].v != 0) { t = i;
        break; }
55    if (t == -1) return poly(n, 0);
56    mint fac = A[t];
57    for (int i = 0; i < n; i++) A[i] /= fac;
58    poly p(A.begin()+t, A.end());
59    p.resize(n);
60    poly q = log(p, n);
61    poly r = exp(q * mint(b), n) * pow(fac, b);
62    if (t == 0) return r;
63    if (b >= n || b*t >= n) return poly(n, 0);
64    r.insert(r.begin(), t*b, mint(0));
65    r.resize(n);
66    return r;
67 } // 8dc32d
68 poly mod(const poly& f, const poly& g) { return quoRem(f,g).
        second; }
69 poly xkmod(ll k, poly f) {
70    poly r{1}, a{0,1};
71    for(;k;k>>=1) {
72        if(k&1) r = mod(conv(r,a), f);
73        a = mod(conv(a,a), f);
74    }
75    return r;
76 } // ef2278
77 // solve recurrence with initial vals s[0], s[1]... s[n-1]
78 // a[k] = c[1]*a[k-1] + c[2]*a[k-2] + ... c[n]*a[k-n]
79 mint solve_linrec(vector<mint> s, vector<mint> c, int n, ll
        k) {
80    poly f(n+1, 0);
81    f[n] = 1;
82    for (int i=0;i<n;i++) f[i] = mint(-c[n-i]);
83    poly r = xkmod(k, f); r.resize(n);
84    mint ans(0);
85    for (int i = 0; i < n; i++) ans += r[i] * mint(s[i]);
86    return ans;
87 } // 902e38
```

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

```
1  typedef vector<double> vd;
2  vd interpolate(vd x, vd y, int n) {
3    vd res(n), temp(n);
4    rep(k,0,n-1) rep(i,k+1,n)
5      y[i] = (y[i] - y[k]) / (x[i] - x[k]);
6    double last = 0; temp[0] = 1;
7    rep(k,0,n) rep(i,0,n) {
8      res[i] += y[k] * temp[i];
9      swap(last, temp[i]);
10     temp[i] -= last * x[k];
11   }
12   return res;
13 }
```

PolyMultipoint.h

Description: Multipoint evaluation and interpolation

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

"PolyOps.h"   20a147, 28 lines

```
1  void segProd(vector<poly>& stor, poly& v, int ind, int l,
        int r) { // v -> places to evaluate at
2    if (l == r) { stor[ind] = {-v[l],1}; return; }
3    int m = (l+r)/2; segProd(stor,v,2*ind,l,m); segProd(stor,v
        ,2*ind+1,m+1,r);
4    stor[ind] = conv(stor[2*ind],stor[2*ind+1]);
5  }
6  void evalAll(vector<poly>& stor, poly& res, poly v, int ind
        = 1) {
7    v = quoRem(v,stor[ind]).second;
```

```
21    if (sz(stor[ind]) == 2) { res.pb(sz(v)?v[0]:0); return; }
22    evalAll(stor,res,v,2*ind); evalAll(stor,res,v,2*ind+1);
23  }
24  // evaluate polynomial v at points in p
25  poly multiEval(poly v, poly p) {
26    vector<poly> stor(4*sz(p)); segProd(stor,p,1,0,sz(p)-1);
27    poly res; evalAll(stor,res,v); return res; }
28
   poly combAll(vector<poly>& stor, poly& dems, int ind, int l,
        int r) {
     if (l == r) return {dems[l]};
     int m = (l+r)/2;
     poly a = combAll(stor,dems,2*ind,l,m), b = combAll(stor,
        dems,2*ind+1,m+1,r);
     return conv(a,stor[2*ind+1])+conv(b,stor[2*ind]);
   }
   poly interpolate(vector<pair<T,T>> v) {
     int n = sz(v); poly x; for (auto t:v) x.pb(t.first);
     vector<poly> stor(4*n); segProd(stor,x,1,0,n-1);
     poly dems; evalAll(stor,dems,dif(stor[1]));
     rep(i,0,n) dems[i] = v[i].second/dems[i];
     return combAll(stor,dems,1,0,n-1);
   }
```

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

```
1  vector<ll> berlekampMassey(vector<ll> s) {
2    int n = sz(s), L = 0, m = 0;
3    vector<ll> C(n), B(n), T;
4    C[0] = B[0] = 1;
5
6    ll b = 1;
7    rep(i,0,n) { ++m;
8      ll d = s[i] % mod;
9      rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
10     if (!d) continue;
11     T = C; ll coef = d * modpow(b, mod-2) % mod;
12     rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
13     if (2 * L > i) continue;
14     L = i + 1 - L; B = T; b = d; m = 0;
15   }
16
17   C.resize(L + 1); C.erase(C.begin());
18   for (ll& x : C) x = (mod - x) % mod;
19   return C;
20 }
```

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

```
1  typedef vector<ll> Poly;
2  ll linearRec(Poly S, Poly tr, ll k) {
3    int n = sz(tr);
4
5    auto combine = [&](Poly a, Poly b) {
6      Poly res(n * 2 + 1);
7      rep(i,0,n+1) rep(j,0,n+1)
8        res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
9      for (int i = 2 * n; i > n; --i) rep(j,0,n)
10       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

```
1  double gss(double a, double b, double (*f)(double)) {
2    double r = (sqrt(5)-1)/2, eps = 1e-7;
3    double x1 = b - r*(b-a), x2 = a + r*(b-a);
4    double f1 = f(x1), f2 = f(x2);
5    while (b-a > eps)
6      if (f1 < f2) { //change to > to find maximum
7        b = x2; x2 = x1; f2 = f1;
8        x1 = b - r*(b-a); f1 = f(x1);
9      } else {
10       a = x1; x1 = x2; f1 = f2;
11       x2 = a + r*(b-a); f2 = f(x2);
12     }
13   return a;
14 }
```

HillClimbing.h

Description: Poor man's optimization for unimodal functions.

8eeeaf, 14 lines

```
1  typedef array<double, 2> P;
2
3  template<class F> pair<double, P> hillClimb(P start, F f) {
4    pair<double, P> cur(f(start), start);
5    for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
6      rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
7        P p = cur.second;
8        p[0] += dx*jmp;
9        p[1] += dy*jmp;
10       cur = min(cur, make_pair(f(p), p));
11     }
12   }
13   return cur;
14 }
```

Integrate.h

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

4756fc, 7 lines

```
1  template<class F>
2  double quad(double a, double b, F f, const int n = 1000) {
3    double h = (b - a) / 2 / n, v = f(a) + f(b);
4    rep(i,1,n*2)
5      v += f(a + i*h) * (i&1 ? 4 : 2);
6    return v * h / 3;
7  }
```

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

```
1  typedef double d;
2  #define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6
3
4  template <class F>
5  d rec(F& f, d a, d b, d eps, d S) {
6    d c = (a + b) / 2;
7    d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
8    if (abs(T - S) <= 15 * eps || b - a < 1e-10)
9      return T + (T - S) / 15;
```

```
10    return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S
         2);
11  }
12  template<class F>
13  d quad(d a, d b, F f, d eps = 1e-8) {
14    return rec(f, a, b, eps, S(a, b));
15  }
```

Simplex.h
Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.
Usage: vvd A = {{1,-1}, {-1,1}, {-1,-2}};
vd b = {1,1,-4}, c = {-1,-1}, x;
T val = LPSolver(A, b, c).solve(x);
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.   aa8530, 68 lines

```
1   typedef double T; // long double, Rational, double + mod<P
        >...
2   typedef vector<T> vd;
3   typedef vector<vd> vvd;
4
5   const T eps = 1e-8, inf = 1/.0;
6   #define MP make_pair
7   #define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s]))
         s=j
8
9   struct LPSolver {
10    int m, n;
11    vi N, B;
12    vvd D;
13
14    LPSolver(const vvd& A, const vd& b, const vd& c) :
15      m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
16        rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
17        rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i
          ];}
18        rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
19        N[n] = -1; D[m+1][n] = 1;
20      }
21
22    void pivot(int r, int s) {
23      T *a = D[r].data(), inv = 1 / a[s];
24      rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
25        T *b = D[i].data(), inv2 = b[s] * inv;
26        rep(j,0,n+2) b[j] -= a[j] * inv2;
27        b[s] = a[s] * inv2;
28      }
29      rep(j,0,n+2) if (j != s) D[r][j] *= inv;
30      rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
31      D[r][s] = inv;
32      swap(B[r], N[s]);
33    }
34
35    bool simplex(int phase) {
36      int x = m + phase - 1;
37      for (;;) {
38        int s = -1;
39        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
40        if (D[x][s] >= -eps) return true;
41        int r = -1;
42        rep(i,0,m) {
43          if (D[i][s] <= eps) continue;
44          if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
45                       < MP(D[r][n+1] / D[r][s], B[r])) r = i;
46        }
47        if (r == -1) return false;
48        pivot(r, s);
49      }
50    }
51
52    T solve(vd &x) {
53      int r = 0;
54      rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
55      if (D[r][n+1] < -eps) {
56        pivot(r, n);
57        if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
58        rep(i,0,m) if (B[i] == -1) {
```

```
59        int s = 0;
60        rep(j,1,n+1) ltj(D[i]);
61        pivot(i, s);
62      }
63    }
64    bool ok = simplex(1); x = vd(n);
65    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
66    return ok ? D[m][n+1] : inf;
67  }
68 };
```

4.3 Matrices

Determinant.h
Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$ bd5cec, 15 lines

```
1   double det(vector<vector<double>>& a) {
2     int n = sz(a); double res = 1;
3     rep(i,0,n) {
4       int b = i;
5       rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
6       if (i != b) swap(a[i], a[b]), res *= -1;
7       res *= a[i][i];
8       if (res == 0) return 0;
9       rep(j,i+1,n) {
10        double v = a[j][i] / a[i][i];
11        if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
12      }
13    }
14    return res;
15  }
```

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

```
1   const ll mod = 12345;
2   ll det(vector<vector<ll>>& a) {
3     int n = sz(a); ll ans = 1;
4     rep(i,0,n) {
5       rep(j,i+1,n) {
6         while (a[j][i] != 0) { // gcd step
7           ll t = a[i][i] / a[j][i];
8           if (t) rep(k,i,n)
9             a[i][k] = (a[i][k] - a[j][k] * t) % mod;
10          swap(a[i], a[j]);
11          ans *= -1;
12        }
13      }
14      ans = ans * a[i][i] % mod;
15      if (!ans) return 0;
16    }
17    return (ans + mod) % mod;
18  }
```

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

```
1   typedef vector<double> vd;
2   const double eps = 1e-12;
3
4   int solveLinear(vector<vd>& A, vd& b, vd& x) {
5     int n = sz(A), m = sz(x), rank = 0, br, bc;
6     if (n) assert(sz(A[0]) == m);
7     vi col(m); iota(all(col), 0);
8
9     rep(i,0,n) {
10      double v, bv = 0;
11      rep(r,i,n) rep(c,i,m)
12        if ((v = fabs(A[r][c])) > bv)
13          br = r, bc = c, bv = v;
14      if (bv <= eps) {
15        rep(j,i,n) if (fabs(b[j]) > eps) return -1;
16        break;
17      }
18      swap(A[i], A[br]);
19      swap(b[i], b[br]);
20      swap(col[i], col[bc]);
```

```
21      rep(j,0,n) swap(A[j][i], A[j][bc]);
22      bv = 1/A[i][i];
23      rep(j,i+1,n) {
24        double fac = A[j][i] * bv;
25        b[j] -= fac * b[i];
26        rep(k,i+1,m) A[j][k] -= fac*A[i][k];
27      }
28      rank++;
29    }
30
31    x.assign(m, 0);
32    for (int i = rank; i--;) {
33      b[i] /= A[i][i];
34      x[col[i]] = b[i];
35      rep(j,0,i) b[j] -= A[j][i] * b[i];
36    }
37    return rank; // (multiple solutions if rank < m)
38  }
```

SolveLinear2.h
Description: To get all uniquely determined values of x back from SolveLinear, make the following changes: 08e495, 7 lines

"SolveLinear.h"
```
1   rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
2   // ... then at the end:
3   x.assign(m, undefined);
4   rep(i,0,rank) {
5     rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
6     x[col[i]] = b[i] / A[i][i];
7   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

```
1   typedef bitset<1000> bs;
2
3   int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
4     int n = sz(A), rank = 0, br;
5     assert(m <= sz(x));
6     vi col(m); iota(all(col), 0);
7     rep(i,0,n) {
8       for (br=i; br<n; ++br) if (A[br].any()) break;
9       if (br == n) {
10        rep(j,i,n) if(b[j]) return -1;
11        break;
12      }
13      int bc = (int)A[br]._Find_next(i-1);
14      swap(A[i], A[br]);
15      swap(b[i], b[br]);
16      swap(col[i], col[bc]);
17      rep(j,0,n) if (A[j][i] != A[j][bc]) {
18        A[j].flip(i); A[j].flip(bc);
19      }
20      rep(j,i+1,n) if (A[j][i]) {
21        b[j] ^= b[i];
22        A[j] ^= A[i];
23      }
24      rank++;
25    }
26
27    x = bs();
28    for (int i = rank; i--;) {
29      if (!b[i]) continue;
30      x[col[i]] = 1;
31      rep(j,0,i) b[j] ^= A[j][i];
32    }
33    return rank; // (multiple solutions if rank < m)
34  }
```

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: $\mathcal{O}(n^3)$ 023944, 38 lines

```
1   using T = double; // or mint
2   #define ABS(x) fabs(x)
3   #define ISZERO(x) (fabs(x) < 1e-12)
```

```
4  // #define ABS(x) (x.v)
5  // #define ISZERO(x) (x.v == 0)
6  int matInv(vector<vector<T>>& A) {
7    int n = sz(A); vi col(n);
8    vector<vector<T>> tmp(n, vector<T>(n));
9    rep(i,0,n) tmp[i][i] = 1, col[i] = i;
10
11   rep(i,0,n) {
12     int r = i, c = i;
13     rep(j,i,n) rep(k,i,n)
14       if (ABS(A[j][k]) > ABS(A[r][c]))
15         r = j, c = k;
16     if (ISZERO(A[r][c])) return i;
17     A[i].swap(A[r]); tmp[i].swap(tmp[r]);
18     rep(j,0,n)
19       swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
20     swap(col[i], col[c]);
21     T v = A[i][i];
22     rep(j,i+1,n) {
23       T f = A[j][i] / v;
24       A[j][i] = 0;
25       rep(k,i+1,n) A[j][k] -= f*A[i][k];
26       rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
27     }
28     rep(j,i+1,n) A[i][j] /= v;
29     rep(j,0,n) tmp[i][j] /= v;
30     A[i][i] = 1;
31   }
32   for (int i = n-1; i > 0; --i) rep(j,0,i) {
33     T v = A[j][i];
34     rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
35   }
36   rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
37   return n;
38 }
```

Gaussian.h
Description: Converts A to rref using Gaussian elim.
Time: $\mathcal{O}\left(N^3\right)$
3dc817, 27 lines

```
1  using T = double; // or mint
2  const double EPS = 1e-9; // adjust?
3  int getRow(vector<vector<double>>& m, int R, int i, int nex)
        {
4    pair<double,int> bes{0,-1}; // find row with max abs value
5    rep(j,nex,R) ckmax(bes,{abs(m[j][i]),j});
6    return bes.first < EPS ? -1 : bes.second; }
7  int getRow(vector<vector<mint>>& m, int R, int i, int nex) {
8    rep(j,nex,R) if (m[j][i] != 0) return j;
9    return -1; }
10 pair<T,int> gauss(vector<vector<T>>& m) { // convert to rref
11   if (!sz(m)) return {1,0};
12   int R = sz(m), C = sz(m[0]), rank = 0, nex = 0;
13   T prod = 1; // determinant
14   rep(i,0,C) {
15     int row = getRow(m,R,i,nex);
16     if (row == -1) { prod = 0; continue; }
17     if (row != nex) prod *= -1, swap(m[row],m[nex]);
18     prod *= m[nex][i]; rank++;
19     T x = 1/m[nex][i]; rep(k,i,C) m[nex][k] *= x;
20     rep(j,0,R) if (j != nex) {
21       T v = m[j][i]; if (v == 0) continue;
22       rep(k,i,C) m[j][k] -= v*m[nex][k];
23     }
24     nex++;
25   }
26   return {prod,rank};
27 }
```

Tridiagonal.h
Description: x = tridiagonal(d, p, q, b) solves the equation system

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

This is useful for solving problems on the type

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

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

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

Fails if the solution is not unique.
If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither tr nor the check for diag[i] == 0 is needed.
Time: $\mathcal{O}(N)$
8f9fa8, 26 lines

```
1  typedef double T;
2  vector<T> tridiagonal(vector<T> diag, const vector<T>& super
        ,
3    const vector<T>& sub, vector<T> b) {
4    int n = sz(b); vi tr(n);
5    rep(i,0,n-1) {
6      if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] ==
            0
7        b[i+1] -= b[i] * diag[i+1] / super[i];
8        if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
9        diag[i+1] = sub[i]; tr[++i] = 1;
10     } else {
11       diag[i+1] -= super[i]*sub[i]/diag[i];
12       b[i+1] -= b[i]*sub[i]/diag[i];
13     }
14   }
15   for (int i = n; i--;) {
16     if (tr[i]) {
17       swap(b[i], b[i-1]);
18       diag[i-1] = diag[i];
19       b[i] /= super[i-1];
20     } else {
21       b[i] /= diag[i];
22       if (i) b[i-1] -= b[i]*super[i-1];
23     }
24   }
25   return b;
26 }
```

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 $\left(\sum a_i^2 + \sum b_i^2\right) \log_2 N < 9 \cdot 10^{14}$ (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

```
1  typedef complex<double> C;
2  typedef vector<double> vd;
3  void fft(vector<C>& a) {
4    int n = sz(a), L = 31 - __builtin_clz(n);
5    static vector<complex<long double>> R(2, 1);
6    static vector<C> rt(2, 1);  // (^ 10% faster if double)
7    for (static int k = 2; k < n; k *= 2) {
8      R.resize(n); rt.resize(n);
9      auto x = polar(1.0L, acos(-1.0L) / k);
10     rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
11   }
12   vi rev(n);
13   rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
14   rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
15   for (int k = 1; k < n; k *= 2)
16     for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
17       C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-
           rolled)
18       a[i + j + k] = a[i + j] - z;
19       a[i + j] += z;
20     }
21 }
22 vd conv(const vd& a, const vd& b) {
23   if (a.empty() || b.empty()) return {};
24   vd res(sz(a) + sz(b) - 1);
25   int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
26   vector<C> in(n), out(n);
27   copy(all(a), begin(in));
28   rep(i,0,sz(b)) in[i].imag(b[i]);
29   fft(in);
30   for (C& x : in) x *= x;
31   rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
```

```
32   fft(out);
33   rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
34   return res;
35 }
```

FFTComplex.h
Description: FFT but with complex numbers
Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim$1s for $N = 2^{22}$)
8cb345, 14 lines

```
1  vector<C> conv_complex(const vector<C>& a, const vector<C>&
        b) {
2    if (a.empty() || b.empty()) return {};
3    vector<C> res(sz(a) + sz(b) - 1);
4    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
5    vector<C> in1(n), in2(n), out(n);
6    copy(all(a), begin(in1));
7    copy(all(b), begin(in2));
8    fft(in1);
9    fft(in2);
10   rep(i,0,n) out[i] = conj(in1[i] * in2[i]);
11   fft(out);
12   rep(i,0,sz(res)) res[i] = conj(out[i]) / C(n, 0);
13   return res;
14 }
```

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, mod).
Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)
"FastFourierTransform.h"
b82773, 22 lines

```
1  typedef vector<ll> vl;
2  template<int M> vl convMod(const vl &a, const vl &b) {
3    if (a.empty() || b.empty()) return {};
4    vl res(sz(a) + sz(b) - 1);
5    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
6    vector<C> L(n), R(n), outs(n), outl(n);
7    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
8    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
9    fft(L), fft(R);
10   rep(i,0,n) {
11     int j = -i & (n - 1);
12     outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
13     outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
14   }
15   fft(outl), fft(outs);
16   rep(i,0,sz(res)) {
17     ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
18     ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
19     res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
20   }
21   return res;
22 }
```

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, mod).
Time: $\mathcal{O}(N \log N)$
"../number-theory/ModPow.h"
ced03d, 33 lines

```
1  const ll mod = (119 << 23) + 1, root = 62; // = 998244353
2  // For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 <<
        21
3  // and 483 << 21 (same root). The last two are > 10^9.
4  typedef vector<ll> vl;
5  void ntt(vl &a) {
6    int n = sz(a), L = 31 - __builtin_clz(n);
7    static vl rt(2, 1);
8    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
9      rt.resize(n);
10     ll z[] = {1, modpow(root, mod >> s)};
11     rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
12   }
13   vi rev(n);
14   rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
15   rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
16   for (int k = 1; k < n; k *= 2)
```

```
17    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
18      ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
19      a[i + j + k] = ai - z + (z > ai ? mod : 0);
20      ai += (ai + z >= mod ? z - mod : z);
21    }
22 }
23 vl conv(const vl &a, const vl &b) {
24    if (a.empty() || b.empty()) return {};
25    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n =
        1 << B;
26    int inv = modpow(n, mod - 2);
27    vl L(a), R(b), out(n);
28    L.resize(n), R.resize(n);
29    ntt(L), ntt(R);
30    rep(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv
        % mod;
31    ntt(out);
32    return {out.begin(), out.begin() + s};
33 }
```

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

```
1 void FST(vi& a, bool inv) {
2    for (int n = sz(a), step = 1; step < n; step *= 2) {
3      for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
4        int &u = a[j], &v = a[j + step]; tie(u, v) =
5          inv ? pii(v - u, u) : pii(v, u + v); // AND
6          inv ? pii(v, u - v) : pii(u + v, u); // OR
7          pii(u + v, u - v);                   // XOR
8      }
9    }
10    if (inv) for (int& x : a) x /= sz(a); // XOR only
11 }
12 vi conv(vi a, vi b) {
13    FST(a, 0); FST(b, 0);
14    rep(i,0,sz(a)) a[i] *= b[i];
15    FST(a, 1); return a;
16 }
```

Number theory (5)

5.1 Modular arithmetic

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

```
1 ll modLog(ll a, ll b, ll m) {
2    ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
3    unordered_map<ll, ll> A;
4    while (j <= n && (e = f = e * a % m) != b % m)
5      A[e * b % m] = j++;
6    if (e == b % m) return j;
7    if (__gcd(m, e) == __gcd(m, b))
8      rep(i,2,n+2) if (A.count(e = e * f % m))
9        return n * i - A[e];
10    return -1;
11 }
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions.
modsum(to, c, k, m) = $\sum_{i=0}^{\text{to}-1} (ki + c)\%m$. divsum is similar but for floored division.
Time: $\log(m)$, with a large constant.
5c5bc5, 16 lines

```
1 typedef unsigned long long ull;
2 ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
3
4 ull divsum(ull to, ull c, ull k, ull m) {
5    ull res = k / m * sumsq(to) + c / m * to;
6    k %= m; c %= m;
7    if (!k) return res;
8    ull to2 = (to * k + c) / m;
9    return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
10 }
```

```
11 ll modsum(ull to, ll c, ll k, ll m) {
12    c = ((c % m) + m) % m;
13    k = ((k % m) + m) % m;
14    return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
15 }
16
```

SiyongModular.h

Description: Modular class
Time: Faster than kactl mod. Slower than using ll directly
67b174, 46 lines

```
1 int const MOD = 998244353;
2 ll euclid(ll a, ll b, ll &x, ll &y) {
3    if (!b) return x = 1, y = 0, a;
4    ll d = euclid(b, a % b, y, x);
5    return y -= a/b * x, d;
6 }
7 struct mint {
8    int v;
9    explicit operator int() {return v;}
10    mint(): v(0) {}
11    mint(auto z) {
12      z %= MOD;
13      if (z < 0) z += MOD;
14      v = z;
15    }
16    friend mint invert(mint a) {
17      ll x, y, g = euclid(a.v, MOD, x, y);
18      assert(g == 1); return mint(x);
19    }
20    mint& operator+= (mint const& o) {
21      if((v+=o.v)>=MOD) v-=MOD; return *this;}
22    mint& operator-= (mint const& o) {
23      if((v-=o.v)<0) v+=MOD; return *this;}
24    mint& operator*= (mint const& o) {
25      v=(ll)v*o.v%MOD; return *this;}
26    mint& operator/= (mint const& o) {
27      return *this *= invert(o);}
28
29    friend mint operator+ (mint a, mint const& b) {
30      return a+=b;}
31    friend mint operator- (mint a, mint const& b) {
32      return a-=b;}
33    friend mint operator* (mint a, mint const& b) {
34      return a*=b;}
35    friend mint operator/ (mint const& a, mint const& b) {
36      return a*invert(b);}
37
38    mint operator- () {return mint(-v);}
39    friend mint pow(mint a, auto b) {
40      mint r(1);
41      for(;b;b>>=1, a*=a)
42        if(b&1)
43          r *= a;
44      return r;
45    }
46 };
```

ModHelpers.h

Description: Computes inv, fact, ifact
Time: $\mathcal{O}(N)$
"SiyongModular.h"
2acc0d, 16 lines

```
1 int const MV = 2e6 + 10;
2 mint inv[MV], fact[MV], ifact[MV];
3 void init() {
4    inv[1] = mint(1);
5    for(int i = 2; i < MV; ++i)
6      inv[i] = mint(MOD - MOD/i) * inv[MOD % i];
7    fact[0] = ifact[0] = mint(1);
8    for(int i = 1; i < MV; ++i) {
9      fact[i] = mint(i) * fact[i-1];
10      ifact[i] = inv[i] * ifact[i-1];
11    }
12 }
13 mint choose(int n, int k) {
14    assert(0 <= k && k <= n); // or return 0
15    return fact[n] * ifact[n-k] * ifact[k];
16 }
```

ModMulLL.h

Description: Calculate $a \cdot b \mod c$ (or $a^b \mod c$) for $0 \le a, b \le c \le 7.2 \cdot 10^{18}$.
Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow
bbbd8f, 11 lines

```
1 typedef unsigned long long ull;
2 ull modmul(ull a, ull b, ull M) {
3    ll ret = a * b - M * ull(1.L / M * a * b);
4    return ret + M * (ret < 0) - M * (ret >= (ll)M);
5 }
6 ull modpow(ull b, ull e, ull mod) {
7    ull ans = 1;
8    for (; e; b = modmul(b, b, mod), e /= 2)
9      if (e & 1) ans = modmul(ans, b, mod);
10    return ans;
11 }
```

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: $\mathcal{O}\left(\log^2 p\right)$ worst case, $\mathcal{O}(\log p)$ for most p
"ModPow.h"
19a793, 24 lines

```
1 ll sqrt(ll a, ll p) {
2    a %= p; if (a < 0) a += p;
3    if (a == 0) return 0;
4    assert(modpow(a, (p-1)/2, p) == 1); // else no solution
5    if (p % 4 == 3) return modpow(a, (p+1)/4, p);
6    // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
7    ll s = p - 1, n = 2;
8    int r = 0, m;
9    while (s % 2 == 0)
10      ++r, s /= 2;
11    while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
12    ll x = modpow(a, (s + 1) / 2, p);
13    ll b = modpow(a, s, p), g = modpow(n, s, p);
14    for (;; r = m) {
15      ll t = b;
16      for (m = 0; m < r && t != 1; ++m)
17        t = t * t % p;
18      if (m == 0) return x;
19      ll gs = modpow(g, 1LL << (r - m - 1), p);
20      g = gs * gs % p;
21      x = x * gs % p;
22      b = b * g % p;
23    }
24 }
```

5.2 Primality

FastEratosthenes.h

Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 1.5s
6b2912, 20 lines

```
1 const int LIM = 1e6;
2 bitset<LIM> isPrime;
3 vi eratosthenes() {
4    const int S = (int)round(sqrt(LIM)), R = LIM / 2;
5    vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)*1.1))
        ;
6    vector<pii> cp;
7    for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
8      cp.push_back({i, i * i / 2});
9      for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
10    }
11    for (int L = 1; L <= R; L += S) {
12      array<bool, S> block{};
13      for (auto &[p, idx] : cp)
14        for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;
15      rep(i,0,min(S, R - L))
16        if (!block[i]) pr.push_back((L + i) * 2 + 1);
17    }
18    for (int i : pr) isPrime[i] = 1;
19    return pr;
20 }
```

PrimeSieve.h

Description: Prime sieve but slow, for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 8.5s
d47ac3, 17 lines

```
1 int const LIM = 1e7+5;
2 vector<bool> cp;
3 vi pr, nx, lp, cnt;
```

```
4  void sieve()
5  {
6    cp.assign(LIM, 0), nx.assign(LIM, -1), lp.assign(LIM, -1),
         cnt.assign(LIM, -1);
7    for(int i=2;i<LIM;++i) {
8      if(!cp[i])
9        lp[i] = pr.size(), nx[i] = cnt[i] = 1, pr.push_back(i)
         ;
10     for(int j=0,k;j<pr.size() && (k=i*pr[j])<LIM;++j) { //
       pr[j]<(LIM+i-1)/i, if there's overflow
11       cp[k] = 1, lp[k] = j;
12       if(j == lp[i]) {
13         nx[k] = nx[i], cnt[k] = cnt[i]+1; break;
14       } else nx[k] = i, cnt[k] = 1;;
15     }
16   }
17 }
```

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
```
1  bool isPrime(ull n) {
2    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
3    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 179526502
       2},
4        s = __builtin_ctzll(n-1), d = n >> s;
5    for (ull a : A) {   // ^ count trailing zeroes
6      ull p = modpow(a%n, d, n), i = s;
7      while (p != 1 && p != n - 1 && a % n && i--)
8        p = modmul(p, p, n);
9      if (p != n-1 && i != s) return 0;
10   }
11   return 1;
12 }
```

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
```
1  ull pollard(ull n) {
2    auto f = [n](ull x) { return modmul(x, x, n) + 1; };
3    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
4    while (t++ % 40 || __gcd(prd, n) == 1) {
5      if (x == y) x = ++i, y = f(x);
6      if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
7      x = f(x), y = f(f(y));
8    }
9    return __gcd(prd, n);
10 }
11 vector<ull> factor(ull n) {
12   if (n == 1) return {};
13   if (isPrime(n)) return {n};
14   ull x = pollard(n);
15   auto l = factor(x), r = factor(n / x);
16   l.insert(l.end(), all(r));
17   return l;
18 }
```

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
```
1  ll euclid(ll a, ll b, ll &x, ll &y) {
2    if (!b) return x = 1, y = 0, a;
3    ll d = euclid(b, a % b, y, x);
4    return y -= a/b * x, d;
5  }
```

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 $0 \le x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.

Time: $\log(n)$
`"euclid.h"`   04d93a, 7 lines
```
1  ll crt(ll a, ll m, ll b, ll n) {
2    if (n > m) swap(a, b), swap(m, n);
3    ll x, y, g = euclid(m, n, x, y);
4    assert((a - b) % g == 0); // else no solution
5    x = (b - a) % n * x % n / g * m + a;
6    return x < 0 ? x + m*n/g : x;
7  }
```

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
```
1  const int LIM = 5000000;
2  int phi[LIM];
3
4  void calculatePhi() {
5    rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
6    for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
7      for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
8  }
```

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
```
1  typedef double d; // for N ~ 1e7; long double for N ~ 1e9
2  pair<ll, ll> approximate(d x, ll N) {
3    ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
4    for (;;) {
5      ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf)
       ,
6          a = (ll)floor(y), b = min(a, lim),
7          NP = b*P + LP, NQ = b*Q + LQ;
8      if (a > b) {
9        // 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);
13     }
14     if (abs(y = 1/(y - (d)a)) > 3*N) {
15       return {NP, NQ};
16     }
17     LP = P; P = NP;
18     LQ = Q; Q = NQ;
19   }
20 }
```

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
```
1  struct Frac { ll p, q; };
2
3  template<class F>
4  Frac fracBS(F f, ll N) {
5    bool dir = 1, A = 1, B = 1;
6    Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
7    if (f(lo)) return lo;
8    assert(f(hi));
9    while (A || B) {
10     ll adv = 0, step = 1; // move hi if dir, else lo
11     for (int si = 0; step; (step *= 2) >>= si) {
12       adv += step;
13       Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
14       if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
15         adv -= step; si = 2;
16       }
17     }
18     hi.p += lo.p * adv;
19     hi.q += lo.q * adv;
20     dir = !dir;
21     swap(lo, hi);
22     A = B; B = !!adv;
23   }
24   return dir ? hi : lo;
25 }
```

5.5 Pythagorean Triples
The Pythagorean triples are uniquely generated by

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

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

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

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

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

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

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

Mobius Inversion:

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

Other useful formulas/forms:

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

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

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

5.9   Lifting the Exponent

For $n > 0$, p prime, and ints x, y s.t. $p \nmid x, y$ and $p \mid x - y$:

- $p \neq 2$ or $p = 2, 4 \mid x - y \implies v_p(x^n - y^n) = v_p(x - y) + v_p(n)$.
- $p = 2, 2 \mid n \implies v_2(x^n - y^n) = v_2((x^2)^{n/2} - (y^2)^{n/2})$.

Combinatorial (6)

6.1   Permutations

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;
}
```

6.1.1   Cycles

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

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

6.1.2   Derangements

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

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

6.1.3   Burnside's lemma

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

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

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

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

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

6.2   Partitions and subsets

6.2.1   Partition function

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

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

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

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

6.2.2   Lucas' Theorem

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

6.2.3   Binomials

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

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k), \ c(0,0) = 1$$

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

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

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.

6.4   Young Tableaux

Let a **Young diagram** have shape $\lambda = (\lambda_1 \geq \cdots \geq \lambda_k)$, where λ_i equals the number of cells in the i-th (left-justified) row from the top. A **Young tableau** of shape λ is a filling of the $n = \sum \lambda_i$ cells with a permutation of $1 \ldots n$ such that each row and column is increasing.

Hook-Length Formula: For the cell in position (i, j), let $h_\lambda(i, j) = |\{(I, J) | i \leq I, j \leq J, (I = i \text{ or } J = j)\}|$. The number of Young tableaux of shape λ is equal to $f^\lambda = \frac{n!}{\prod h_\lambda(i,j)}$.

Schensted's Algorithm: converts a permutation σ of length n into a pair of Young Tableaux $(S(\sigma), T(\sigma))$ of the same shape. When inserting $x = \sigma_i$,

1. Add x to the first row of S by inserting x in place of the largest y with $x < y$. If y doesn't exist, push x to the end of the row, set the value of T at that position to be i, and stop.

2. Add y to the second row using the same rule, keep repeating as necessary.

For $\sigma = (5, 2, 3, 1, 4)$,

$$S(\sigma): \ 5 \ \rightarrow \ \begin{array}{cc} 2 \\ 5 \end{array} \ \rightarrow \ \begin{array}{cc} 2 & 3 \\ 5 \end{array} \ \rightarrow \ \begin{array}{cc} 1 & 3 \\ 2 \\ 5 \end{array} \ \rightarrow \ \begin{array}{ccc} 1 & 3 & 4 \\ 2 \\ 5 \end{array}$$

$$T(\sigma): \ 1 \ \rightarrow \ \begin{array}{cc} 1 \\ 2 \end{array} \ \rightarrow \ \begin{array}{cc} 1 & 3 \\ 2 \end{array} \ \rightarrow \ \begin{array}{cc} 1 & 3 \\ 2 \\ 4 \end{array} \ \rightarrow \ \begin{array}{ccc} 1 & 3 & 5 \\ 2 \\ 4 \end{array}$$

All pairs $(S(\sigma), T(\sigma))$ of the same shape correspond to a unique σ, so $n! = \sum (f^\lambda)^2$. Also, $S(\sigma^R) = S(\sigma)^T$.

Let $d_k(\sigma), a_k(\sigma)$ be the lengths of the longest subseqs which are a union of k decreasing/ascending subseqs, respectively. Then $a_k(\sigma) = \sum_{i=1}^{k} \lambda_i, d_k(\sigma) = \sum_{i=1}^{k} \lambda_i^*$, where λ_i^* is size of the i-th column.

6.4.1 Prufer Sequences

Tree to seq: Take lowest label leaf, print label of neighbour, until 2 nodes left

Seq to tree: deg of each node = frequency in sequence+1. take lowest 1-degree node connect it to first value in sequence, increase sequence pointer. repeat.

6.5 Other

DeBruijnSeq.h
Description: Given alphabet $[0, k)$ constructs a cyclic string of length k^n that contains every length n string as substr.
2dfa89, 13 lines

```
vi deBruijnSeq(int k, int n) {
  if (k == 1) return {0};
  vi seq, aux(n+1);
  function<void(int,int)> gen = [&](int t, int p) {
    if (t > n) { // +1yndon word of len p
      if (n%p == 0) rep(i,1,p+1) seq.pb(aux[i]);
    } else {
      aux[t] = aux[t-p]; gen(t+1,p);
      while (++aux[t] < k) gen(t+1,t);
    }
  };
  gen(1,1); return seq;
}
```

NimProduct.h
Description: Product of nimbers is associative, commutative, and distributive over addition (xor). Forms finite field of size 2^{2^k}. Defined by $ab = \text{mex}(\{a'b + ab' + a'b' : a' < a, b' < b\})$. Application: Given 1D coin turning games G_1, G_2 $G_1 \times G_2$ is the 2D coin turning game defined as follows. If turning coins at $x_1, x_2, \ldots, x_m$ is legal in G_1 and $y_1, y_2, \ldots, y_n$ is legal in G_2, then turning coins at all positions (x_i, y_j) is legal assuming that the coin at (x_m, y_n) goes from heads to tails. Then the grundy function $g(x, y)$ of $G_1 \times G_2$ is $g_1(x) \times g_2(y)$.
Time: 64^2 xors per multiplication, memorize to speed up.
6f6dc4, 45 lines

```
using ul = uint64_t;
struct Precalc {
  ul tmp[64][64], y[8][8][256];
  unsigned char x[256][256];
  Precalc() { // small nim products, all < 256
    rep(i,0,256) rep(j,0,256) x[i][j] = mult<8>(i,j);
    rep(i,0,8) rep(j,0,i+1) rep(k,0,256)
      y[i][j][k] = mult<64>(prod2(8*i,8*j),k);
  }
  ul prod2(int i, int j) { // nim prod of 2^i, 2^j
    ul& u = tmp[i][j]; if (u) return u;
    if (!(i&j)) return u = 1ULL<<(i|j);
    int a = (i&j)&-(i&j); // a=2^k, consider 2^{2^k}
    return u=prod2(i^a,j)^prod2((i^a)|(a-1),(j^a)|(i&(a-1)));
    ; // 2^{2^k}*2^{2^k} = 2^{2^k}+2^{2^k-1}
  } // 2^{2^i}*2^{2^j} = 2^{2^i+2^j} if i<j
  template<int L> ul mult(ul a, ul b) {
    ul c = 0; rep(i,0,L) if (a>>i&1)
      rep(j,0,L) if (b>>j&1) c ^= prod2(i,j);
    return c;
  }
  // 2^{8*i}*(a>>(8*i)&255) * 2^{8*j}*(b>>(8*j)&255)
  // -> (2^{8*i}*2^{8*j})*((a>>(8*i)&255)*(b>>(8*j)&255))
  ul multFast(ul a, ul b) const { // faster nim product
    ul res = 0; auto f=[](ul c,int d) {return c>>(8*d)&255
      ;};
    rep(i,0,8) {
      F0R(j,i) res ^= y[i][j][x[f(a,i)][f(b,j)]]
        ^x[f(a,j)][f(b,i)]];
      res ^= y[i][i][x[f(a,i)][f(b,i)]];
    }
    return res;
  }
};
const Precalc P;
```

```
struct nb { // nimber
  ul x; nb() { x = 0; }
  nb(ul _x): x(_x) {}
  explicit operator ul() { return x; }
  nb operator+(nb y) { return nb(x^y.x); }
  nb operator*(nb y) { return nb(P.multFast(x,y.x)); }
  friend nb pow(nb b, ul p) {
    nb res = 1; for (;p;p/=2,b=b*b) if (p&1) res = res*b;
    return res; } // b^{2^{2^A}-1}=1 where 2^{2^A} > b
  friend nb inv(nb b) { return pow(b,-2); }
};
```

Graph (7)

7.1 Network flow

MinCostMaxFlow.h
Description: Min-cost max-flow. All capacities are 0. Flows are initialized to be negative.
Time: Originally $\mathcal{O}\left(E^2\right)$
c2fbd7, 66 lines

```
// #include <bits/extc++.h>
struct MCMF {
  typedef int C; typedef int F; typedef ll R;
  C const INFC = numeric_limits<C>::max() / 4;
  F const INFF = numeric_limits<F>::max() / 4;

  struct Edge {int n; F flow; C cost; size_t rev;};
  int N;
  vector<vector<Edge>> ed;
  vector<C> dist, pi;
  vector<F> amt;
  vector<size_t> par;

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

  void addEdge(int u, int v, F f, C c) {
    ed[u].emplace_back(v, -f, c, ed[v].size());
    ed[v].emplace_back(u, 0, -c, ed[u].size()-1);
  }

  void path(int s) { // 417ab0
    fill(all(amt), 0); amt[s] = INFF;
    fill(all(dist), INFC); dist[s] = 0;

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

    while (!q.empty()) {
      s = q.top().second; q.pop();
      C di = dist[s] + pi[s];
      for (auto [n, f, c, rev]: ed[s])
        if (f < 0 && ckmin(dist[n], di + c - pi[n])) {
          par[n] = rev; amt[n] = min(amt[s], -f);
          if(its[n] == q.end()) its[n] = q.push({-dist[n], n
        });
          else q.modify(its[n], {-dist[n], n});
        }
    }
    rep(i,0,N) pi[i] = amt[i] ? pi[i] + dist[i] : INFC;
  }

  pair<F, R> maxflow(int s, int t) { // 2126d0
    F totflow = 0; R totcost = 0;
    while (path(s), amt[t]) {
      F fl = amt[t]; totflow += fl;
      for(int n = t;n != s;) {
        auto &[p, f, c, rev] = ed[n][par[n]];
        f -= fl; ed[p][rev].flow += fl; n = p;
        totcost -= (R) fl * c;//OR +=(R)fl*ed[p][rev].cost
      }
    }
    return {totflow, totcost};
  }

  // If some costs can be negative, call this before maxflow
  // :
  void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INFC); pi[s] = 0;
```

```
    int it = N, ch = 1;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INFC)
        for (auto [to, f, c, _]: ed[i])
          if (f < 0 && ckmin(pi[to], pi[i] + c)) ch = 1;
    assert(it >= 0); // negative cost cycle
  }
}; // 0b54fa without setpi; 88d7c5 with setpi
```

NetworkSimplex.h
Description: Network simplex algorithm
Time: empircally fast
6909e1, 67 lines

```
using i128 = __int128_t;
struct NetworkSimplex {
  using Flow = int64_t; using Cost = int64_t; using V_id =
    int32_t; using E_id = int32_t;
  struct Edge { V_id src, dst; Flow flow, cap; Cost cost; };
  Cost INF = 1; i128 delta = 0;
  int n; vector<Flow> B; vector<Cost> P; vector<Edge> E;
  vi pei, depth; vector<set<int>> tree;
  void init(int _n) { n = _n; B.resize(n+1); pei.assign(n+1
    ,-1);
    depth.resize(n+1); P.resize(n+1); tree.resize(n+1); }
  int ae(V_id a, V_id b, Flow l, Flow u, Cost c) {
    E.pb({a,b,0,u-l,c}); E.pb({b,a,0,0,-c});
    delta += l*c; B[b] += l, B[a] -= l;
    return sz(E)-2;
  }
  void upd(E_id ei) {
    P[E[ei].dst] = P[E[ei].src]+E[ei].cost;
    pei[E[ei].dst] = ei^1;
    depth[E[ei].dst] = 1+depth[E[ei].src];
    dfs(E[ei].dst);
  }
  void dfs(int node) { for(auto& ei:tree[node]) if (ei !=
    pei[node]) upd(ei); }
  // applies cb to a -> b and (tree path b -> a)
  template<class CB> void walk(int ei, CB cb) {
    cb(ei);
    for (V_id a = E[ei].src, b = E[ei].dst; a != b; ) {
      if (depth[a] > depth[b]) cb(pei[a]^1), a = E[pei[a]].
    dst;
      else cb(pei[b]), b = E[pei[b]].dst;
    }
  }
  i128 solve() {
    const int m = sz(E);
    for (E_id i = 0; i < m; i += 2) INF += abs(E[i].cost);
    rep(i,0,n) {
      V_id a = n, b = i; Cost c = B[i];
      if (c < 0) c *= -1, swap(a,b);
      E_id ei = ae(a,b,0,c,-INF);
      tree[a].insert(ei), tree[b].insert(ei^1);
    }
    dfs(n);
    i128 answer = delta;
    E_id ptr = 0;
    const int BLOCK = n/3+1;
    rep(z,0,sz(E)/BLOCK+1) {
      pair<Cost,E_id> pin{0,-1};
      for (int t = 0; t < BLOCK; ++t, (++ptr) %= sz(E)) {
        const auto& e = E[ptr];
        if (e.flow < e.cap) ckmin(pin, mp(e.cost+P[e.src]-P[
    e.dst],ptr));
      }
      auto [cost, ein] = pin;
      if (cost == 0) continue;
      pair<Cost,E_id> pout{E[ein].cap-E[ein].flow, ein};
      walk(ein,[&](E_id ei) { ckmin(pout, mp(E[ei].cap-E[ei
    ].flow,ei)); });
      auto [flow, eout] = pout;
      walk(ein,[&](E_id ei) { E[ei].flow += flow, E[ei^1].
    flow -= flow; });
      tree[E[ein].src].insert(ein), tree[E[ein].dst].insert(
    ein^1);
      tree[E[eout].src].erase(eout), tree[E[eout].dst].erase
    (eout^1);
      upd(pei[E[eout].src] == eout ? ein : ein^1);
      answer += i128(flow)*cost; // why can't this loop?
      z = -1;
    }
    rep(i,0,n) {
```

```
62    if (E[m+2*i].flow < E[m+2*i].cap) throw 5;
63    answer += i128(E[m+2*i].flow)*INF;
64  }
65  return answer;
66  }
67 };
```

Dinic.h
Description: Dinic's without scaling
99f97c, 64 lines

```
1 struct Edge {
2   int u, v;
3   ll cap, flow;
4   Edge() {}
5   Edge(int u, int v, ll cap): u(u), v(v), cap(cap), flow(0)
        {}
6 };
7 struct Dinic {
8   int N;
9   vector<Edge> E;
10  vector<vector<int>> g;
11  vector<int> d, pt;
12  Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}
13  void AddEdge(int u, int v, ll cap) {
14    if (u != v) {
15      E.emplace_back(u, v, cap);
16      g[u].emplace_back(E.size() - 1);
17      E.emplace_back(v, u, 0);
18      g[v].emplace_back(E.size() - 1);
19    }
20  }
21  bool BFS(int S, int T) {
22    queue<int> q({S});
23    fill(d.begin(), d.end(), N + 1);
24    d[S] = 0;
25    while (!q.empty()) {
26      int u = q.front(); q.pop();
27      if (u == T) break;
28      for (int k: g[u]) {
29        Edge &e = E[k];
30        if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
31          d[e.v] = d[e.u] + 1;
32          q.emplace(e.v);
33        }
34      }
35    }
36    return d[T] != N + 1;
37  }
38  ll DFS(int u, int T, ll flow = -1) {
39    if (u == T || flow == 0) return flow;
40    for (int &i = pt[u]; i < g[u].size(); ++i) {
41      Edge &e = E[g[u][i]];
42      Edge &oe = E[g[u][i]^1];
43      if (d[e.v] == d[e.u] + 1) {
44        ll amt = e.cap - e.flow;
45        if (flow != -1 && amt > flow) amt = flow;
46        if (ll pushed = DFS(e.v, T, amt)) {
47          e.flow += pushed;
48          oe.flow -= pushed;
49          return pushed;
50        }
51      }
52    }
53    return 0;
54  }
55  ll MaxFlow(int S, int T) {
56    ll total = 0;
57    while (BFS(S, T)) {
58      fill(pt.begin(), pt.end(), 0);
59      while (ll flow = DFS(S, T))
60        total += flow;
61    }
62    return total;
63  }
64 };
```

DinicWithScaling.h
Description: Flow algorithm with complexity $O(VE \log U)$ where $U = \max |cap|$. $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.
d7f0f1, 42 lines

```
1 struct Dinic {
2   struct Edge {
```

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

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

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

```
1 pair<int, vi> globalMinCut(vector<vi> mat) {
2   pair<int, vi> best = {INT_MAX, {}};
3   int n = sz(mat);
4   vector<vi> co(n);
5   rep(i,0,n) co[i] = {i};
6   rep(ph,1,n) {
7     vi w = mat[0];
8     size_t s = 0, t = 0;
9     rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio.
        queue
10      w[t] = INT_MIN;
11      s = t, t = max_element(all(w)) - w.begin();
12      rep(i,0,n) w[i] += mat[t][i];
13    }
14    best = min(best, {w[t] - mat[t][t], co[t]});
15    co[s].insert(co[s].end(), all(co[t]));
16    rep(i,0,n) mat[s][i] += mat[t][i];
17    rep(i,0,n) mat[i][s] = mat[s][i];
18    mat[0][t] = INT_MIN;
19  }
20  return best;
21 }
```

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

Time: $O(V)$ Flow Computations
"PushRelabel.h"
0418b3, 13 lines

```
1 typedef array<ll, 3> Edge;
2 vector<Edge> gomoryHu(int N, vector<Edge> ed) {
3   vector<Edge> tree;
4   vi par(N);
5   rep(i,1,N) {
6     PushRelabel D(N); // Dinic also works
7     for (Edge t: ed) D.addEdge(t[0], t[1], t[2], t[2]);
8     tree.push_back({i, par[i], D.calc(i, par[i])});
9     rep(j,i+1,N)
10      if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
11  }
12  return tree;
13 }
```

7.2 Matching

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

```
1 bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
2   if (btoa[j] == -1) return 1;
3   vis[j] = 1; int di = btoa[j];
4   for (int e: g[di])
5     if (!vis[e] && find(e, g, btoa, vis)) {
6       btoa[e] = di;
7       return 1;
8     }
9   return 0;
10 }
11 int dfsMatching(vector<vi>& g, vi& btoa) {
12   vi vis;
13   rep(i,0,sz(g)) {
14     vis.assign(sz(btoa), 0);
15     for (int j: g[i])
16       if (find(j, g, btoa, vis)) {
17         btoa[j] = i;
18         break;
19       }
20   }
21   return sz(btoa) - (int)count(all(btoa), -1);
22 }
```

MinimumVertexCover.h
Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.
"DFSMatching.h"
da4196, 20 lines

```
1 vi cover(vector<vi>& g, int n, int m) {
2   vi match(m, -1);
3   int res = dfsMatching(g, match);
4   vector<bool> lfound(n, true), seen(m);
5   for (int it: match) if (it != -1) lfound[it] = false;
6   vi q, cover;
7   rep(i,0,n) if (lfound[i]) q.push_back(i);
8   while (!q.empty()) {
9     int i = q.back(); q.pop_back();
10    lfound[i] = 1;
11    for (int e: g[i]) if (!seen[e] && match[e] != -1) {
12      seen[e] = true;
13      q.push_back(match[e]);
14    }
15  }
16  rep(i,0,n) if (!lfound[i]) cover.push_back(i);
17  rep(i,0,m) if (seen[i]) cover.push_back(n+i);
18  assert(sz(cover) == res);
19  return cover;
20 }
```

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][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
}
```

GeneralMatchingBlossom.h

Description: Variant on Gabow's Impl of Edmond's Blossom Algorithm. General unweighted max matching with 1-based indexing. If white[v] = 0 after solve() returns, v is part of every max matching. 1-INDEXED NODES

Time: $\mathcal{O}(NM)$, faster in practice

6dfcb8, 53 lines

```
using vb = vector<bool>;
using pi = pair<int,int>;
using vpi = vector<pi>;
struct MaxMatching {
  int N; vector<vi> adj;
  vector<int> mate, first; vb white; vpi label;
  void init(int _N) { N = _N; adj = vector<vi>(N+1);
    mate = first = vi(N+1); label = vpi(N+1); white = vb(N+1
    ); }
  void ae(int u, int v) { adj.at(u).pb(v), adj.at(v).pb(u);
    }
  int group(int x) { if (white[first[x]]) first[x] = group(
    first[x]);
    return first[x]; }
  void match(int p, int b) {
    swap(b,mate[p]); if (mate[b] != p) return;
    if (!label[p].second) mate[b] = label[p].first, match(
      label[p].first,b); // vertex label
    else match(label[p].first,label[p].second), match(label[
      p].second,label[p].first); // edge label
  }
  bool augment(int st) { assert(st);
    white[st] = 1; first[st] = 0; label[st] = {0,0};
    queue<int> q; q.push(st);
    while (!q.empty()) {
      int a = q.front(); q.pop(); // outer vertex
      for(auto &b:adj[a]) { assert(b);
        if (white[b]) { // two outer vertices, form blossom
          int x = group(a), y = group(b), lca = 0;
          while (x||y) {
            if (y) swap(x,y);
            if (label[x] == pi{a,b}) { lca = x; break; }
            label[x] = {a,b}, x = group(label[mate[x]].first
            );
          }
          for (int v: {group(a),group(b)}) while (v != lca)
          {
            assert(!white[v]); // make everything along path
            white
            q.push(v); white[v] = true; first[v] = lca;
            v = group(label[mate[v]].first);
          }
        } else if (!mate[b]) { // found augmenting path
          mate[b] = a; match(a,b); white = vb(N+1); // reset
          return true;
        } else if (!white[mate[b]]) {
          white[mate[b]] = true; first[mate[b]] = b;
          label[b] = {0,0}; label[mate[b]] = pi{a,0};
          q.push(mate[b]);
        }
      }
    }
    return false;
  }
  int solve() {
    int ans = 0; // mate pairs node with its match,
      bidirectional
    rep(st,1,N+1) if (!mate[st]) ans += augment(st);
    rep(st,1,N+1) if (!mate[st] && !white[st]) assert(!
      augment(st));
    return ans;
  }
};
```

GeneralWeightedMatching.h

Description: General max weight max matching with 1-based indexing. Edge weights must be positive, combo of UnweightedMatch and Hungarian. 1-INDEXED NODES

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

fe0333, 145 lines

```
template<int SZ> struct WeightedMatch {
  struct edge { int u,v,w; }; edge g[SZ*2][SZ*2];
  void ae(int u, int v, int w) { g[u][v].w = g[v][u].w = w;
  int N,NX,lab[SZ*2],match[SZ*2],slack[SZ*2],st[SZ*2];
  int par[SZ*2],floFrom[SZ*2][SZ],S[SZ*2],aux[SZ*2];
  vi flo[SZ*2]; queue<int> q;
  void init(int _N) { N = _N; // init all edges
    rep(u,1,N+1) rep(v,1,N+1) g[u][v] = {u,v,0}; }
  int eDelta(edge e) { // >= 0 at all times
    return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2; }
  void updSlack(int u, int x) { // smallest edge -> blossom
    x
    if (!slack[x] || eDelta(g[u][x]) < eDelta(g[slack[x]][x
    ]))
      slack[x] = u; }
  void setSlack(int x) {
    slack[x] = 0; rep(u,1,N+1) if (g[u][x].w > 0
    && st[u] != x && S[st[u]] == 0) updSlack(u,x); }
  void qPush(int x) {
    if (x <= N) q.push(x);
    else for(auto &t:flo[x]) qPush(t); }
  void setSt(int x, int b) {
    st[x] = b; if (x > N) for(auto &t:flo[x]) setSt(t,b); }
  int getPr(int b, int xr) { // get even position of xr
    int pr = find(all(flo[b]),xr)-begin(flo[b]);
    if (pr&1) { reverse(1+all(flo[b])); return sz(flo[b])-pr
    ; }
    return pr; }
  void setMatch(int u, int v) { // rearrange flo[u], matches
    edge e = g[u][v]; match[u] = e.v; if (u <= N) return;
    int xr = floFrom[u][e.u], pr = getPr(u,xr);
    rep(i,0,pr) setMatch(flo[u][i],flo[u][i^1]);
    setMatch(xr,v); rotate(begin(flo[u]),pr+all(flo[u])); }
  void augment(int u, int v) { // set matches including u->v
    while (1) { // and previous ones
      int xnv = st[match[u]]; setMatch(u,v);
      if (!xnv) return;
      setMatch(xnv,st[par[xnv]]);
      u = st[par[xnv]], v = xnv;
    }
  }
  int lca(int u, int v) { // same as in unweighted
    static int t = 0; // except maybe return 0
    for (++t;u||v;swap(u,v)) {
      if (!u) continue;
      if (aux[u] == t) return u;
      aux[u] = t; u = st[match[u]];
      if (u) u = st[par[u]];
    }
    return 0;
  }
  void addBlossom(int u, int anc, int v) {
    int b = N+1; while (b <= NX && st[b]) ++b;
    if (b > NX) ++NX; // new blossom
```

```
    lab[b] = S[b] = 0; match[b] = match[anc]; flo[b] = {anc
    };
    auto blossom = [&](int x) {
      for (int y; x != anc; x = st[par[y]])
        flo[b].pb(x), flo[b].pb(y = st[match[x]]), qPush(y);
    };
    blossom(u); reverse(1+all(flo[b])); blossom(v); setSt(b,
    b);
    // identify all nodes in current blossom
    rep(x,1,NX+1) g[b][x].w = g[x][b].w = 0;
    rep(x,1,N+1) floFrom[b][x] = 0;
    for(auto &xs:flo[b]) { // find tightest constraints
      rep(x,1,NX+1) if (g[b][x].w == 0 || eDelta(g[xs][x]) <
        eDelta(g[b][x])) g[b][x]=g[xs][x], g[x][b]=g[x][xs];
      rep(x,1,N+1) if (floFrom[xs][x]) floFrom[b][x] = xs;
    } // floFrom to deconstruct blossom
    setSlack(b); // since didn't qPush everything
  }
  void expandBlossom(int b) {
    for(auto &t:flo[b]) setSt(t,t); // undo setSt(b,b)
    int xr = floFrom[b][g[b][par[b]].u], pr = getPr(b,xr);
    for(int i = 0; i < pr; i += 2) {
      int xs = flo[b][i], xns = flo[b][i+1];
      par[xs] = g[xns][xs].u; S[xs] = 1; // no setSlack(xns)
      ?
      S[xns] = slack[xs] = slack[xns] = 0; qPush(xns);
    }
    S[xr] = 1, par[xr] = par[b];
    rep(i,pr+1,sz(flo[b])) { // matches don't change
      int xs = flo[b][i]; S[xs] = -1, setSlack(xs); }
    st[b] = 0; // blossom killed
  }
  bool onFoundEdge(edge e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) { // v unvisited, matched with smth else
      par[v] = e.u, S[v] = 1; slack[v] = 0;
      int nu = st[match[v]]; S[nu] = slack[nu] = 0; qPush(nu
      );
    } else if (S[v] == 0) {
      int anc = lca(u,v); // if 0 then match found!
      if (!anc) return augment(u,v),augment(v,u),1;
      addBlossom(u,anc,v);
    }
    return 0;
  }
  bool matching() {
    q = queue<int>();
    rep(x,1,NX+1) {
      S[x] = -1, slack[x] = 0; // all initially unvisited
      if (st[x] == x && !match[x]) par[x] = S[x] = 0, qPush(
      x);
    }
    if (!sz(q)) return 0;
    while (1) {
      while (sz(q)) { // unweighted matching with tight
        edges
        int u = q.front(); q.pop(); if (S[st[u]] == 1)
      continue;
        rep(v,1,N+1) if (g[u][v].w > 0 && st[u] != st[v]) {
          if (eDelta(g[u][v]) == 0) { // condition is strict
            if (onFoundEdge(g[u][v])) return 1;
          } else updSlack(u,st[v]);
        }
      }
      int d = INT_MAX;
      rep(b,N+1,NX+1) if (st[b] == b && S[b] == 1)
        ckmin(d,lab[b]/2); // decrease lab[b]
      rep(x,1,NX+1) if (st[x] == x && slack[x]) {
        if (S[x] == -1) ckmin(d,eDelta(g[slack[x]][x]));
        else if (S[x] == 0) ckmin(d,eDelta(g[slack[x]][x])/2
      );
      } // edge weights shouldn't go below 0
      rep(u,1,N+1) {
        if (S[st[u]] == 0) {
          if (lab[u] <= d) return 0; // why?
          lab[u] -= d;
        } else if (S[st[u]] == 1) lab[u] += d;
      } // lab has opposite meaning for verts and blossoms
      rep(b,N+1,NX+1) if (st[b] == b && S[b] != -1)
        lab[b] += (S[b] == 0 ? 1 : -1)*d*2;
      q = queue<int>();
      rep(x,1,NX+1) if (st[x]==x && slack[x] // new tight
      edge
```

```
26        && st[slack[x]] != x && eDelta(g[slack[x]][x]) == 0)
27        if (onFoundEdge(g[slack[x]][x])) return 1;
28    rep(b,N+1,NX+1) if (st[b]==b && S[b]==1 && lab[b]==0)
29        expandBlossom(b); // odd dist blossom taken apart
30    }
31    return 0;
32  }
33  pair<ll,int> calc() {
34    NX = N; st[0] = 0; rep(i,1,2*N+1) aux[i] = 0;
35    rep(i,1,N+1) match[i] = 0, st[i] = i, flo[i].clear();
36    int wMax = 0;
37    rep(u,1,N+1) rep(v,1,N+1)
38      floFrom[u][v] = (u == v ? u : 0), ckmax(wMax,g[u][v].w2
39      );
40    rep(u,1,N+1) lab[u] = wMax; // start high and decrease
41    int num = 0; ll wei = 0; while (matching()) ++num;
42    rep(u,1,N+1) if (match[u] && match[u] < u)
43      wei += g[u][match[u]].w; // edges in matching
44    return {wei,num};
45  }
46 };
```

7.3 DFS algorithms

SCC.h
Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: scc(graph, [&](vi& v) { ... }) visits all components in reverse topological order. comp[i] holds the component index of a node (a component only has edges to components with lower index). ncomps will contain the number of components.
Time: $\mathcal{O}(E + V)$
<div align="right">76b5c9, 24 lines</div>

```
1  vi val, comp, z, cont;
2  int Time, ncomps;
3  template<class G, class F> int dfs(int j, G& g, F& f) {
4    int low = val[j] = ++Time, x; z.push_back(j);
5    for (auto e : g[j]) if (comp[e] < 0)
6      low = min(low, val[e] ?: dfs(e,g,f));
7
8    if (low == val[j]) {
9      do {
10        x = z.back(); z.pop_back();
11        comp[x] = ncomps;
12        cont.push_back(x);
13      } while (x != j);
14      f(cont); cont.clear();
15      ncomps++;
16    }
17    return val[j] = low;
18  }
19  template<class G, class F> void scc(G& g, F f) {
20    int n = sz(g);
21    val.assign(n, 0); comp.assign(n, -1);
22    Time = ncomps = 0;
23    rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
24  }
```

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: $\mathcal{O}(E + V)$
<div align="right">2965e5, 33 lines</div>

```
1  vi num, st;
2  vector<vector<pii>> ed;
3  int Time;
4  template<class F>
5  int dfs(int at, int par, F& f) {
6    int me = num[at] = ++Time, e, y, top = me;
7    for (auto pa : ed[at]) if (pa.second != par) {
8      tie(y, e) = pa;
9      if (num[y]) {
10        top = min(top, num[y]);
11        if (num[y] < me)
```

```
12        st.push_back(e);
13      } else {
14        int si = sz(st);
15        int up = dfs(y, e, f);
16        top = min(top, up);
17        if (up == me) {
18          st.push_back(e);
19          f(vi(st.begin() + si, st.end()));
20          st.resize(si);
21        }
22        else if (up < me) st.push_back(e);
23        else { /* e is a bridge */ }
24      }
25    }
26    return top;
27  }
28
29  template<class F>
30  void bicomps(F f) {
31    num.assign(sz(ed), 0);
32    rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
33  }
```

BlockCutTree.h
Description: Builds the block cut tree. BCTree node n is an AP if $n >= cut$, and it corresponds to node who[n][0] in original graph. Node v of the original graph is an AP if $vmap[v] >= cut$. $emap[i] = -1$ if edge i is a bridge. Otherwise, $emap[i]$ is the BCC containing it.
Usage: see BiconnectedComponents.h
edges[i] = the edge i (pair of two nodes)
Time: $\mathcal{O}(E + V)$
"BiconnectedComponents.h"
<div align="right">7343ca, 49 lines</div>

```
1  vector<pii> edges;
2  tuple<int, vector<vi>, vector<vi>, vi, vi> BCTree() {
3    int N = ed.size(), M = edges.size();
4    vector<int> emap(M, -1); // edge -> bicomp id
5    vector<vi> bclist; // list of biconnected components
6    bicomps([&](vector<int> &&eds) {
7      for(int x: eds) emap[x] = bclist.size();
8      bclist.emplace_back(eds);
9    });
10
11    vector<int> vmap(N, -1);
12    for(int i = 0;i < M; ++i)
13      if(emap[i] == -1) { // bridge: connects two APs
14        auto [u, v] = edges[i];
15        vmap[u] = vmap[v] = -2;
16      }
17
18    for(int i = 0;i < bclist.size();++i)
19      for(int x: bclist[i]) {
20        auto [u, v] = edges[x];
21        for(int j = 2;j--; swap(u, v))
22          if(vmap[u] == -1) vmap[u] = i;
23          else if(vmap[u] != i) vmap[u] = -2;
24      }
25
26    int const cut = bclist.size();
27    int TN = bclist.size();
28    vector<vi> who(TN);
29    for(int i = 0;i < N;++i)
30      if(vmap[i] == -2) vmap[i] = TN++, who.emplace_back(1, i)
31      ; else who[vmap[i]].emplace_back(i);
32    vector<vi> tadj(TN);
33
34    for(int i = 0;i < N; ++i)
35      if(cut <= vmap[i]) // if 'i' is an AP
36        for(auto [x, e]: ed[i]) {
37          if(emap[e] == -1) // Bridge: connect both APs
38            tadj[vmap[i]].push_back(vmap[x]);
39          else {
40            tadj[vmap[i]].push_back(emap[e]);
41            tadj[emap[e]].push_back(vmap[i]);
42          }
43        }
44    for(auto &v: tadj) { // one AP can connect to a BCC in
45      multiple ways
46      sort(all(v));
47      v.resize(distance(v.begin(), unique(all(v))));
48    }
49    return {cut, tadj, who, emap, vmap};
```

2sat.h
Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a\|\|b)\&\&(!a\|\|c)\&\&(d\|\|!b)\&\&...$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions ($\sim x$).
Usage: TwoSat ts(number of boolean variables);
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.setValue(2); // Var 2 is true
ts.atMostOne({0,~1,2}); // <= 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.
<div align="right">5f9706, 56 lines</div>

```
1  struct TwoSat {
2    int N;
3    vector<vi> gr;
4    vi values; // 0 = false, 1 = true
5
6    TwoSat(int n = 0) : N(n), gr(2*n) {}
7
8    int addVar() { // (optional)
9      gr.emplace_back();
10      gr.emplace_back();
11      return N++;
12    }
13
14    void either(int f, int j) {
15      f = max(2*f, -1-2*f);
16      j = max(2*j, -1-2*j);
17      gr[f].push_back(j^1);
18      gr[j].push_back(f^1);
19    }
20    void setValue(int x) { either(x, x); }
21
22    void atMostOne(const vi& li) { // (optional)
23      if (sz(li) <= 1) return;
24      int cur = ~li[0];
25      rep(i,2,sz(li)) {
26        int next = addVar();
27        either(cur, ~li[i]);
28        either(cur, next);
29        either(~li[i], next);
30        cur = ~next;
31      }
32      either(cur, ~li[1]);
33    }
34
35    vi val, comp, z; int time = 0;
36    int dfs(int i) {
37      int low = val[i] = ++time, x; z.push_back(i);
38      for(int e : gr[i]) if (!comp[e])
39        low = min(low, val[e] ?: dfs(e));
40      if (low == val[i]) do {
41        x = z.back(); z.pop_back();
42        comp[x] = low;
43        if (values[x>>1] == -1)
44          values[x>>1] = x&1;
45      } while (x != i);
46      return val[i] = low;
47    }
48
49    bool solve() {
50      values.assign(N, -1);
51      val.assign(2*N, 0); comp = val;
52      rep(i,0,2*N) if (!comp[i]) dfs(i);
53      rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
54      return 1;
55    }
56 };
```

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: $\mathcal{O}(V + E)$
<div align="right">780b64, 15 lines</div>

```
1  vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0)
2  {
```

```
2    int n = sz(gr);
3    vi D(n), its(n), eu(nedges), ret, s = {src};
4    D[src]++; // to allow Euler paths, not just cycles
5    while (!s.empty()) {
6      int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
7      if (it == end){ ret.push_back(x); s.pop_back(); continue
8        ; }
9      tie(y, e) = gr[x][it++];
10     if (!eu[e]) {
11       D[x]--, D[y]++;
12       eu[e] = 1; s.push_back(y);
13     }}
14   for (int x : D) if (x < 0 || sz(ret) != nedges+1) return
15     {};
16   return {ret.rbegin(), ret.rend()};
17 }
```

BipolarOrientation.h

Description: Finds a bipolar orientation of a biconnected graph

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

adc2bf, 35 lines

```
1  // Warning: Mutates the vector 'a'
2  vector<int> bipolarOrient(vector<vector<int> > a, int s, int
    t) {
3    size_t N = a.size(); // must have s != t, N >= 2
4    vector<int> o(N), p(N, -1), d(N, -1), l(N), lk[2];
5    lk[0]=lk[1]=vector<int>(N,-1); // lk[0] = prev, lk[1] =
      next
6    a[s].insert(a[s].begin(), t); // can duplicate edge
7    int time=0;
8    auto f=[&](auto& f, int n) ->void{
9      o[time]=n,l[n]=d[n]=time++;
10     for(int x:a[n]) if(x!=p[n])
11     {
12       if(d[x]==-1) {
13         p[x]=n, f(f, x); // assert(n==s || l[x]<d[n]);
14         ckmin(l[n], l[x]);
15       } else ckmin(l[n], d[x]);
16     }
17   };
18   f(f, s);
19   auto add=[&](int u, int v, bool b){
20     lk[!b][v]=lk[!b][u]; // b true: before, b false: after
21     lk[!b][u]=v;
22     lk[b][v]=u;
23     if(lk[!b][v]!=-1) lk[b][lk[!b][v]]=v;
24   };
25   add(s, t, 0);
26   vector<char> sgn(N, 0);
27   sgn[t]=1;
28   for(int i=2;i<N;++i) {
29     int n=o[i];
30     add(p[n], n, sgn[p[n]]=!sgn[o[l[n]]]);
31   } // assert(lk[0][s] == -1);
32   vector<int> ans;
33   for(;s!=-1;s=lk[1][s]) ans.push_back(s);
34   return ans;
35 }
```

7.4 Coloring

EdgeColoring.h

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

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

e210e2, 31 lines

```
1  vi edgeColoring(int N, vector<pii> eds) {
2    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
3    for (pii e : eds) ++cc[e.first], ++cc[e.second];
4    int u, v, ncols = *max_element(all(cc)) + 1;
5    vector<vi> adj(N, vi(ncols, -1));
6    for (pii e : eds) {
7      tie(u, v) = e;
8      fan[0] = v;
9      loc.assign(ncols, 0);
10     int at = u, end = u, d, c = free[u], ind = 0, i = 0;
11     while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
12       loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
13     cc[loc[d]] = c;
14     for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd
        ])
```

```
15     swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
16     while (adj[fan[i]][d] != -1) {
17       int left = fan[i], right = fan[++i], e = cc[i];
18       adj[u][e] = left;
19       adj[left][e] = u;
20       adj[right][e] = -1;
21       free[right] = e;
22     }
23     adj[u][d] = fan[i];
24     adj[fan[i]][d] = u;
25     for (int y : {fan[0], u, end})
26       for (int& z = free[y] = 0; adj[y][z] != -1; z++);
27   }
28   rep(i,0,sz(eds))
29     for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
30   return ret;
31 }
```

7.5 Heuristics

MaximalCliques.h

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

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

b0d5b1, 12 lines

```
1  typedef bitset<128> B;
2  template<class F>
3  void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R
    ={}) {
4    if (!P.any()) { if (!X.any()) f(R); return; }
5    auto q = (P | X)._Find_first();
6    auto cands = P & ~eds[q];
7    rep(i,0,sz(eds)) if (cands[i]) {
8      R[i] = 1;
9      cliques(eds, f, P & eds[i], X & eds[i], R);
10     R[i] = P[i] = 0; X[i] = 1;
11   }
12 }
```

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

```
1  typedef vector<bitset<200>> vb;
2  struct Maxclique {
3    double limit=0.025, pk=0;
4    struct Vertex { int i, d=0; };
5    typedef vector<Vertex> vv;
6    vb e;
7    vv V;
8    vector<vi> C;
9    vi qmax, q, S, old;
10   void init(vv& r) {
11     for (auto& v : r) v.d = 0;
12     for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
13     sort(all(r), [](auto a, auto b) { return a.d > b.d; });
14     int mxD = r[0].d;
15     rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
16   }
17   void expand(vv& R, int lev = 1) {
18     S[lev] += S[lev - 1] - old[lev];
19     old[lev] = S[lev - 1];
20     while (sz(R)) {
21       if (sz(q) + R.back().d <= sz(qmax)) return;
22       q.push_back(R.back().i);
23       vv T;
24       for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i
         });
25       if (sz(T)) {
26         if (S[lev]++ / ++pk < limit) init(T);
27         int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1,
          1);
28         C[1].clear(), C[2].clear();
29         for (auto v : T) {
30           int k = 1;
31           auto f = [&](int i) { return e[v.i][i]; };
32           while (any_of(all(C[k]), f)) k++;
33           if (k > mxk) mxk = k, C[mxk + 1].clear();
34           if (k < mnk) T[j++].i = v.i;
```

```
35         C[k].push_back(v.i);
36       }
37       if (j > 0) T[j - 1].d = 0;
38       rep(k,mnk,mxk + 1) for (int i : C[k])
39         T[j].i = i, T[j++].d = k;
40       expand(T, lev + 1);
41     } else if (sz(q) > sz(qmax)) qmax = q;
42     q.pop_back(), R.pop_back();
43   }
44 }
45 vi maxClique() { init(V), expand(V); return qmax; }
46 Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S)
47 {
48   rep(i,0,sz(e)) V.push_back({i});
49 }
50 };
```

MaximumIndependentSet.h

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

7.6 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

```
1  struct LCA {
2    int T = 0;
3    vi time, path, ret;
4    RMQ<int> rmq;
5
6    LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret))
       {}
7    void dfs(vector<vi>& C, int v, int par) {
8      time[v] = T++;
9      for (int y : C[v]) if (y != par) {
10       path.push_back(v), ret.push_back(time[v]);
11       dfs(C, y, v);
12     }
13   }
14
15   int lca(int a, int b) {
16     if (a == b) return a;
17     tie(a, b) = minmax(time[a], time[b]);
18     return path[rmq.query(a, b)];
19   }
20   //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)
       ];}
21 };
```

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

```
1  typedef vector<pair<int, int>> vpi;
2  vpi compressTree(LCA& lca, const vi& subset) {
3    static vi rev; rev.resize(sz(lca.time));
4    vi li = subset, &T = lca.time;
5    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
6    sort(all(li), cmp);
7    int m = sz(li)-1;
8    rep(i,0,m) {
9      int a = li[i], b = li[i+1];
10     li.push_back(lca.lca(a, b));
11   }
12   sort(all(li), cmp);
13   li.erase(unique(all(li)), li.end());
14   rep(i,0,sz(li)) rev[li[i]] = i;
15   vpi ret = {pii(0, li[0])};
16   rep(i,0,sz(li)-1) {
17     int a = li[i], b = li[i+1];
18     ret.emplace_back(rev[lca.lca(a, b)], b);
19   }
20   return ret;
```

```
21 }
```

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

```
1  template <bool VALS_EDGES> struct HLD {
2    int N, tim = 0;
3    vector<vi> adj;
4    vi par, siz, depth, rt, pos;
5    Node *tree;
6    HLD(vector<vi> adj_)
7      : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1), depth(N
         ),
8        rt(N),pos(N),tree(new Node(0, N)){ dfsSz(0); dfsHld(0)
         ; }
9    void dfsSz(int v) {
10     if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v])
         );
11     for (int& u : adj[v]) {
12       par[u] = v, depth[u] = depth[v] + 1;
13       dfsSz(u);
14       siz[v] += siz[u];
15       if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
16     }
17   }
18   void dfsHld(int v) {
19     pos[v] = tim++;
20     for (int u : adj[v]) {
21       rt[u] = (u == adj[v][0] ? rt[v] : u);
22       dfsHld(u);
23     }
24   }
25   template <class B> void process(int u, int v, B op) {
26     for (; rt[u] != rt[v]; v = par[rt[v]]) {
27       if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
28       op(pos[rt[v]], pos[v] + 1);
29     }
30     if (depth[u] > depth[v]) swap(u, v);
31     op(pos[u] + VALS_EDGES, pos[v] + 1);
32   }
33   void modifyPath(int u, int v, int val) {
34     process(u, v, [&](int l, int r) { tree->add(l, r, val);
         });
35   }
36   int queryPath(int u, int v) { // Modify depending on
         problem
37     int res = -1e9;
38     process(u, v, [&](int l, int r) {
39       res = max(res, tree->query(l, r));
40     });
41     return res;
42   }
43   int querySubtree(int v) { // modifySubtree is similar
44     return tree->query(pos[v] + VALS_EDGES, pos[v] + siz[v])
         ;
45   }
46 };
```

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

```
1  struct Edge { int a, b; ll w; };
2  struct Node {
3    Edge key;
4    Node *l, *r;
5    ll delta;
6    void prop() {
7      key.w += delta;
8      if (l) l->delta += delta;
9      if (r) r->delta += delta;
10     delta = 0;
11   }
```

```
12   Edge top() { prop(); return key; }
13 };
14 Node *merge(Node *a, Node *b) {
15   if (!a || !b) return a ?: b;
16   a->prop(), b->prop();
17   if (a->key.w > b->key.w) swap(a, b);
18   swap(a->l, (a->r = merge(b, a->r)));
19   return a;
20 }
21 void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }
22
23 pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
24   RollbackUF uf(n);
25   vector<Node*> heap(n);
26   for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e})
       ;
27   ll res = 0;
28   vi seen(n, -1), path(n), par(n);
29   seen[r] = r;
30   vector<Edge> Q(n), in(n, {-1,-1}), comp;
31   deque<tuple<int, int, vector<Edge>>> cycs;
32   rep(s,0,n) {
33     int u = s, qi = 0, w;
34     while (seen[u] < 0) {
35       if (!heap[u]) return {-1,{}};
36       Edge e = heap[u]->top();
37       heap[u]->delta -= e.w, pop(heap[u]);
38       Q[qi] = e, path[qi++] = u, seen[u] = s;
39       res += e.w, u = uf.find(e.a);
40       if (seen[u] == s) {
41         Node* cyc = 0;
42         int end = qi, time = uf.time();
43         do cyc = merge(cyc, heap[w = path[--qi]]);
44         while (uf.join(u, w));
45         u = uf.find(u), heap[u] = cyc, seen[u] = -1;
46         cycs.push_front({u, time, {&Q[qi], &Q[end]}});
47       }
48     }
49     rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
50   }
51
52   for (auto& [u,t,comp] : cycs) { // restore sol (optional)
53     uf.rollback(t);
54     Edge inEdge = in[u];
55     for (auto& e : comp) in[uf.find(e.b)] = e;
56     in[uf.find(inEdge.b)] = inEdge;
57   }
58   rep(i,0,n) par[i] = in[i].a;
59   return {res, par};
60 }
```

Centroid.h

Description: Boilerplate centroid decomp code

Time: $\mathcal{O}(N \log N)$ db481b, 32 lines

```
1  struct Centroid {
2    vector<vi> const& adj;
3    int N;
4    vector<int> s, rem;
5    vector<vi> links;
6    vector<pair<int, int> > par; // <parent centroid, index>
7    int root;
8    int dfs(int n, int p=-1) {
9      s[n]=1;
10     for(int x: adj[n]) if(x!=p && !rem[x])
11       s[n]+=dfs(x,n);
12     return s[n];
13   }
14   int find(int n, int t, int p=-1) {
15     for(int k=1;k--;)
16       for(int x:adj[n]) if(x!=p && !rem[x] && s[x]*2>t)
17         {p=n,n=x,k=1; break;}
18     return n;
19   }
20   int cent(int start=0) {
21     int c = find(start, dfs(start));
22     // Do stuff with c. Just remember to check both (x != p
          && !rem[x])
23     rem[c]=1;
24     for(int x:adj[c]) if(!rem[x]) {
25       int v = cent(x);
26       par[v] = {c, sz(links[c])};
27       links[c].push_back(v);
```

```
28     }
29     return c;
30   }
31   Centroid(vector<vi> const& adj): adj(adj), N(adj.size()),
32     s(N), rem(N), links(N), par(N,{-1,-1}), root(cent()) {}
33 };
```

JacobLinkCut.h

Description: Link-cut tree with evert, node update and path sum.

Time: All operations take amortized $\mathcal{O}(\log N)$.   ae6b6c, 80 lines

```
1  struct node {
2    node *p,*c[2];
3    ll v, subv;
4    int rev;
5    node():p(NULL),rev(0),v(0),subv(0){c[0]=c[1]=NULL;}
6    node(node *_p):p(_p),rev(0),v(0),subv(0){c[0]=c[1]=NULL;}
7    int state() {
8      if (!p) return -1;
9      if (this==p->c[0]) return 0;
10     else if (this==p->c[1]) return 1;
11     return -1;
12   }
13   bool isroot() { return state()==-1; }
14   node* prop() { // propagate rev, pushdown
15     if (rev) {
16       rev=0;
17       swap(c[0],c[1]);
18       if (c[0]) c[0]->rev^=1;
19       if (c[1]) c[1]->rev^=1;
20     }
21     return this;
22   }
23   void set() { // update any subtrees, pullup
24     subv=v;
25     if (c[0]) subv+=c[0]->subv;
26     if (c[1]) subv+=c[1]->subv;
27   }
28   void res() {
29     if (p->p) p->p->prop();
30     p->prop(); prop();
31   }
32   friend void linknode(node *px, node *x, int d) {
33     if (px && d!=-1) px->c[d] = x;
34     if (x) x->p = px;
35   }
36   void rot() {
37     int d=state(),d2=p->state();
38     node *b=c[!d],*pp=p,*gp=p->p;
39     linknode(pp,b,d);
40     linknode(this,pp,!d);
41     linknode(gp,this,d2);
42     c[!d]->set(); set();
43   }
44   node* splay() {
45     while (!isroot()) {
46       res();
47       if (p->isroot()) rot();
48       else if (state()==p->state()) p->rot(),rot();
49       else rot(),rot();
50     }
51     return prop();
52   }
53   node* find_min() {
54     node *x=this;
55     while(x->prop()->c[0]) x=x->c[0];
56     return x->splay();
57   }
58 };
59 void access(node *x) {
60   node *prev=NULL;
61   for (node *z=x;z;z=z->p) {
62     z->splay();
63     z->c[1]=prev;
64     z->set();
65     prev=z;
66   }
67   x->splay();
68 }
69 void evert(node *x) {access(x); x->rev^=1;}
70 void link(node *x,node *y) {
71   evert(x); access(x); access(y);
72   y->c[1]=x; x->p=y; y->set();
```

```
73  }
74  void cut(node *x,node *y) {
75    evert(y); access(x); access(y);
76    x->p=NULL;
77  }
78  node* find_root(node *x) {access(x); return x->find_min();}
79  void update(node *x, ll v) {access(x); x->v += v; x->splay()
       }
80  ll query(node *x, node *y) {evert(x); access(y); return y->
       subv;}
```

Geometry (8)

8.1   Geometric primitives

Point.h
Description: Class to handle points in the plane. T can be e.g. double or
long long. (Avoid int.)
                                                         47ec0a, 28 lines

```
1   template <class T> int sgn(T x) { return (x > 0) - (x < 0);
        }
2   template<class T>
3   struct Point {
4     typedef Point P;
5     T x, y;
6     explicit Point(T x=0, T y=0) : x(x), y(y) {}
7     bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y)
        ; }
8     bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y)
        ; }
9     P operator+(P p) const { return P(x+p.x, y+p.y); }
10    P operator-(P p) const { return P(x-p.x, y-p.y); }
11    P operator*(T d) const { return P(x*d, y*d); }
12    P operator/(T d) const { return P(x/d, y/d); }
13    T dot(P p) const { return x*p.x + y*p.y; }
14    T cross(P p) const { return x*p.y - y*p.x; }
15    T cross(P a, P b) const { return (a-*this).cross(b-*this);
        }
16    T dist2() const { return x*x + y*y; }
17    double dist() const { return sqrt((double)dist2()); }
18    // angle to x-axis in interval [-pi, pi]
19    double angle() const { return atan2(y, x); }
20    P unit() const { return *this/dist(); } // makes dist()=1
21    P perp() const { return P(-y, x); } // rotates +90 degrees
22    P normal() const { return perp().unit(); }
23    // returns point rotated 'a' radians ccw around the origin
24    P rotate(double a) const {
25      return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
26    friend ostream& operator<<(ostream& os, P p) {
27      return os << "(" << p.x << "," << p.y << ")"; }
28  };
```

lineDistance.h
Description:
Returns the signed distance between point p and the line
containing points a and b. Positive value on left side and
negative on right as seen from a towards b. a==b gives
nan. P is supposed to be Point<T> or Point3D<T>
where T is e.g. double or long long. It uses products in
intermediate steps so watch out for overflow if using int or
long long. Using Point3D will always give a non-negative
distance. For Point3D, call .dist on the result of the cross
product.
                                   f6bf6b, 4 lines

```
1   template<class P>
2   double lineDist(const P& a, const P& b, const P& p) {
3     return (double)(b-a).cross(p-a)/(b-a).dist();
4   }
```

SegmentDistance.h
Description:
Returns the shortest distance between point p and the line
segment from s to e.
Usage: Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
"Point.h"
                                   5c88f4, 6 lines

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

```
5     return ((p-s)*d-(e-s)*t).dist()/d;
6   }
```

SegmentIntersection.h
Description:
If a unique intersection point between the line segments
going from s1 to e1 and from s2 to e2 exists then it is
returned. If no intersection point exists an empty vector
is returned. If infinitely many exist a vector with 2 ele-
ments is returned, containing the endpoints of the com-
mon line segment. The wrong position will be returned if
P is Point<ll> and the intersection point does not have in-
teger coordinates. Products of three coordinates are used
in intermediate steps so watch out for overflow if using int
or long long.
Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
"Point.h", "OnSegment.h"
                               9d57f2, 13 lines

```
1   template<class P> vector<P> segInter(P a, P b, P c, P d) {
2     auto oa = c.cross(d, a), ob = c.cross(d, b),
3          oc = a.cross(b, c), od = a.cross(b, d);
4     // Checks if intersection is single non-endpoint point.
5     if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
6       return {(a * ob - b * oa) / (ob - oa)};
7     set<P> s;
8     if (onSegment(c, d, a)) s.insert(a);
9     if (onSegment(c, d, b)) s.insert(b);
10    if (onSegment(a, b, c)) s.insert(c);
11    if (onSegment(a, b, d)) s.insert(d);
12    return {all(s)};
13  }
```

lineIntersection.h
Description:
If a unique intersection point of the lines going through
s1,e1 and s2,e2 exists {1, point} is returned. If no inter-
section point exists {0, (0,0)} is returned and if infinitely
many exists {-1, (0,0)} is returned. The wrong position
will be returned if P is Point<ll> and the intersection
point does not have integer coordinates. Products of three
coordinates are used in intermediate steps so watch out for
overflow if using int or ll.
Usage: auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;
"Point.h"
                               a01f81, 8 lines

```
1   template<class P>
2   pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
3     auto d = (e1 - s1).cross(e2 - s2);
4     if (d == 0) // if parallel
5       return {-(s1.cross(e1, s2) == 0), P(0, 0)};
6     auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
7     return {1, (s1 * p + e1 * q) / d};
8   }
```

sideOf.h
Description: Returns where p is as seen from s towards e. 1/0/-1 ⇔ left/on
line/right. If the optional argument eps is given 0 is returned if p is within
distance eps from the line. P is supposed to be Point<T> where T is e.g.
double or long long. It uses products in intermediate steps so watch out for
overflow if using int or long long.
Usage: bool left = sideOf(p1,p2,q)==1;
"Point.h"
                               3af81c, 9 lines

```
1   template<class P>
2   int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }
3
4   template<class P>
5   int sideOf(const P& s, const P& e, const P& p, double eps) {
6     auto a = (e-s).cross(p-s);
7     double l = (e-s).dist()*eps;
8     return (a > l) - (a < -l);
9   }
```

OnSegment.h
Description: Returns true iff p lies on the line segment from s to e. Use
(segDist(s,e,p)<=epsilon) instead when using Point<double>.
"Point.h"
                               c597e8, 3 lines

```
1   template<class P> bool onSegment(P s, P e, P p) {
2     return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
```

linearTransformation.h
Description:
Apply the linear transformation (translation, rotation and
scaling) which takes line p0-p1 to line q0-q1 to point r.
"Point.h"
                               03a306, 6 lines

```
1   typedef Point<double> P;
2   P linearTransformation(const P& p0, const P& p1,
3       const P& q0, const P& q1, const P& r) {
4     P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
5     return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2
        ();
6   }
```

LineProjRefl.h
Description: Projects point p onto line ab. Set refl=true to get reflection
of point p across line ab insted. The wrong point will be returned if P is an
integer point and the desired point doesn't have integer coordinates. Products
of three coordinates are used in intermediate steps so watch out for overflow.
"Point.h"
                               b5562d, 5 lines

```
1   template<class P>
2   P lineProj(P a, P b, P p, bool refl=false) {
3     P v = b - a;
4     return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
5   }
```

Angle.h
Description: A class for ordering angles (as represented by int points and a
number of rotations around the origin). Useful for rotational sweeping. Some-
times also represents points or vectors.
Usage: vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of positively
oriented triangles with vertices at 0 and i
                               0f0602, 31 lines

```
1   struct Angle {
2     int x, y;
3     int t;
4     Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
5     Angle operator-(Angle b) const { return {x-b.x, y-b.y, t};
        }
6     int half() const { assert(x || y); return y < 0 || (y == 0
         && x < 0); }
7     Angle t90() const { return {-y, x, t + (half() && x >= 0)
        }; }
8     Angle t180() const { return {-x, -y, t + half()}; }
9     Angle t360() const { return {x, y, t + 1}; }
10  };
11  bool operator<(Angle a, Angle b) {
12    // add a.dist2() and b.dist2() to also compare distances
13    return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
14           make_tuple(b.t, b.half(), a.x * (ll)b.y);
15  }
16  // Given two points, this calculates the smallest angle
        between
17  // them, i.e., the angle that covers the defined line
        segment.
18  pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
19    if (b < a) swap(a, b);
20    return (b < a.t180() ?
21           make_pair(a, b) : make_pair(b, a.t360()));
22  }
23  Angle operator+(Angle a, Angle b) { // point a + vector b
24    Angle r(a.x + b.x, a.y + b.y, a.t);
25    if (a.t180() < r) r.t--;
26    return r.t180() < a ? r.t360() : r;
27  }
28  Angle angleDiff(Angle a, Angle b) { // angle b - angle a
29    int tu = b.t - a.t; a.t = b.t;
30    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)
        };
31  }
```

8.2   Circles

CircleIntersection.h
Description: Computes the pair of points at which two circles intersect. Re-
turns false in case of no intersection.
"Point.h"
                               84d6d3, 10 lines

```cpp
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out
    ) {
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d
    2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d
    2);
  *out = {mid + per, mid - per};
  return true; }
```

CircleTangents.h
Description: Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents − 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.
"Point.h"   b0153d, 12 lines
```cpp
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2
    ) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out; }
```

CircleLine.h
Description: Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.
"Point.h"   e0cfba, 7 lines
```cpp
template<class P> vector<P> circleLine(P c, double r, P a, P
    b) {
  P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
  double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  P h = ab.unit() * sqrt(h2);
  return {p - h, p + h}; }
```
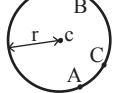
CirclePolyIntersection.h
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$
"../../content/geometry/Point.h"   a1ee63, 18 lines
```cpp
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2
        ();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det)
        );
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum; }
```

circumcircle.h
Description:

ccRadius = radius and ccCenter = center of circle through points A, B, C.

"Point.h"   1caa3a, 7 lines
```cpp
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
    abs((B-A).cross(C-A))/2; }
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h
Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$
"circumcircle.h"   09dd0a, 17 lines
```cpp
pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

8.3   Polygons

InsidePolygon.h
Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.
Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
bool in = inPolygon(v, P{3, 3}, false);
Time: $\mathcal{O}(n)$
"Point.h", "OnSegment.h", "SegmentDistance.h"   2bf504, 11 lines
```cpp
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0
        ;
  }
  return cnt;
}
```

PolygonArea.h
Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!
"Point.h"   f12300, 6 lines
```cpp
template<class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

PolygonCenter.h
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$
"Point.h"   9706dc, 9 lines
```cpp
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```
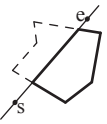
PolygonCut.h
Description:
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));
"Point.h", "lineIntersection.h"   f2b7d4, 13 lines


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

PolygonUnion.h
Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)
Time: $\mathcal{O}(N^2)$, where N is the total number of points
"Point.h", "sideOf.h"   3931c6, 33 lines
```cpp
typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y;
    }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i != j) {
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])
            ];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
            segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>
    0){
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.
        0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

ConvexHull.h
Description:
Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Time: $\mathcal{O}(n \log n)$
"Point.h"   310954, 13 lines

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

HullDiameter.h
Description: Returns the two points with max distance on a convex hull
(ccw, no duplicate/collinear points).
Time: $\mathcal{O}(n)$

"Point.h"   c571b8, 12 lines

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

PointInsideHull.h
Description: Determine whether a point t lies inside a convex hull (CCW
order, with no collinear points). Returns true if point lies within the hull. If
strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

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

```cpp
typedef Point<ll> P;

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

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

"Point.h"   7cf45b, 39 lines

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

```cpp
  }
  return lo;
}

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

HalfPlaneIntersection.h
Description: Returns vertices of half-plane intersection. A half-plane is the
area to the left of a ray, which is defined by a point p and a direction dp.
Area of intersection should be sufficiently precise when all inputs are integers
with magnitude $\leq 10^5$. Intersection must be bounded. Probably works with
floating point too (but EPS might need to be adjusted?).
Time: $\mathcal{O}(N \log N)$

"Point.h"   2040ff, 61 lines

```cpp
using P = Point<double>;
using vP = vector<P>;
const double EPS = 1e-9; // adjust as needed

// -1 if lower half, 0 if origin, 1 if upper half, needs to
 be int type
int half(P x) { return x.y != 0 ? sgn(x.y) : -sgn(x.x); }
bool angleCmp(P a, P b) { int A = half(a), B = half(b);
  return A == B ? a.cross(b) > 0 : A < B; }

struct Ray {
  P p, dp; // origin, direction
  P isect(const Ray& L) const {
    return p+dp*(L.dp.cross(L.p-p)/L.dp.cross(dp)); }
  bool operator<(const Ray& L) const {
    return angleCmp(dp,L.dp); }
};

vP halfPlaneIsect(vector<Ray> rays, bool add_bounds = false)
 {
  if (add_bounds) { // bound input by rectangle [0,DX] x [0,
DY]
    int DX = 1e9, DY = 1e9;
    rays.pb({P{0,0},P{1,0}});
    rays.pb({P{DX,0},P{0,1}});
    rays.pb({P{DX,DY},P{-1,0}});
    rays.pb({P{0,DY},P{0,-1}});
  }
  sort(all(rays)); // sort rays by angle
  { // remove parallel rays
    vector<Ray> nrays;
    for(auto& t : rays) {
      if (!sz(nrays) || nrays.back().dp.cross(t.dp) > EPS)
{ nrays.pb(t); continue; }
      // last two rays are parallel, keep only one
      if (t.dp.cross(t.p-nrays.back().p) > 0) nrays.back() =
 t;
    }
    swap(rays, nrays);
  }
  auto bad = [&](const Ray& a, const Ray& b, const Ray& c) {
    P p1 = a.isect(b), p2 = b.isect(c);
    if ((p2-p1).dot(b.dp) <= EPS) {
```

```cpp
      if (a.dp.cross(c.dp) <= 0) return 2; // isect(a,b,c) =
 empty
      return 1; // isect(a,c) == isect(a,b,c)
    }
    return 0; // all three rays matter
  };
#define reduce(t) \
  while (sz(poly) > 1) { \
    int b = bad(poly.at(sz(poly)-2),poly.back(),t); \
    if (b == 2) return {}; \
    if (b == 1) poly.pop_back(); \
    else break; \
  }
  deque<Ray> poly;
  for(auto& t: rays) { reduce(t); poly.pb(t); }
  for(;;poly.pop_front()) {
    reduce(poly[0]);
    if (!bad(poly.back(),poly[0],poly[1])) break;
  }
  assert(sz(poly) >= 3); // expect nonzero area
  vP poly_points; rep(i,0,sz(poly))
    poly_points.pb(poly[i].isect(poly[(i+1)%sz(poly)]));
  return poly_points;
}
```

MinkowskiSum.h
Description: Minkowski sum of two convex polygons given in CCW order.
example: diameter of hull A is max in A+(-A)
Time: $\mathcal{O}(N)$

    9de734, 19 lines

```cpp
using vP = vector<P>;
vP minkowski_sum(vP a, vP b) {
  if (sz(a) > sz(b)) swap(a, b);
  if (!sz(a)) return {};
  if (sz(a) == 1) {for (auto &t: b) t += a.front(); return b
;}
  rotate(begin(a), min_element(all(a)), end(a));
  rotate(begin(b), min_element(all(b)), end(b));
  a.pb(a[0]), a.pb(a[1]);
  b.pb(b[0]), b.pb(b[1]);
  vP result;
  int i = 0, j = 0;
  while (i < sz(a)-2 || j < sz(b)-2) {
    result.pb(a[i]+b[j]);
    T crs = (a[i+1]-a[i]).cross(b[j+1]-b[j]);
    i += (crs >= 0);
    j += (crs <= 0);
  }
  return result;
}
```

8.4   Misc. Point Set Problems

ClosestPair.h
Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

"Point.h"   ac41a6, 17 lines

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

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

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

```
"Point.h"                                                    df6f59, 23 lines
typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
  vi id(sz(ps));
  iota(all(id), 0);
  vector<array<int, 3>> edges;
  rep(k,0,4) {
    sort(all(id), [&](int i, int j) {
        return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
    map<int, int> sweep;
    for (int i : id) {
      for (auto it = sweep.lower_bound(-ps[i].y);
           it != sweep.end(); sweep.erase(it++)) {
        int j = it->second;
        P d = ps[i] - ps[j];
        if (d.y > d.x) break;
        edges.push_back({d.y + d.x, i, j});
      }
      sweep[-ps[i].y] = i;
    }
    for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.
        y);
  }
  return edges;
}
```

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

```
"Point.h"                                                    bac5b0, 63 lines
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

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

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

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

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

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

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

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
```
```
    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }

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

Delaunay.h
Description: Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points.
Time: $\mathcal{O}(hull3d)$

```
                                                            14d818, 16 lines
using P = Point<ll>;
vector<array<P,3>> triHull(vector<P> p) {
  vector<P3> p3; vector<array<P,3>> res; for (auto &x:p) p3.
      pb(P3{x.x,x.y,x.dist2()});
  bool ok = 0; for (auto &t:p3) ok |= !coplanar(p3[0],p3[1],
      p3[2],t);
  if (!ok) { // all points concyclic
    sort(1+all(p),[&p](P a, P b) {
      return (a-p.front()).cross(b-p.front())>0; });
    rep(i,1,sz(p)-1) res.pb({p.front(),p[i],p[i+1]});
  } else {
    #define nor(z) P(p3[z].x,p3[z].y)
    for(auto &t:hull3dFast(p3))
      if (cross(p3[t[0]],p3[t[1]],p3[t[2]]).dot(P3{0,0,1}) <
          0)
        res.pb({nor(t[0]),nor(t[2]),nor(t[1])});
  }
  return res;
}
```

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

```
"Point.h"                                                    eefdf5, 87 lines
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

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

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B >
      0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r
      ();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
```
```
}
pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back())
        ;
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

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

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

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

8.5 3D

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

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

Point3D.h
Description: Class to handle points in 3D space. T can be e.g. double or long long.

```
                                                            8058ae, 32 lines
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z)
      {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
```

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

3dHull-template-benq.h
Description: 3d hull helpers
481285, 36 lines

```cpp
using T = double;
using P3 = Point3D<double>;
using vb = vector<bool>;

mt19937 rng;
P3 cross(const P3& a, const P3& b, const P3& c) {
  return (b-a).cross(c-a); }
P3 perp(const P3& a, const P3& b, const P3& c) {
  return cross(a,b,c).unit(); }
bool isMult(const P3& a, const P3& b) { // for long longs
  P3 c = a.cross(b);
  return (c.x== 0 && c.y==0 && c.z==0);
}
bool collinear(const P3& a, const P3& b, const P3& c) {
  return isMult(b-a,c-a); }
T DC(const P3&a,const P3&b,const P3&c,const P3&p) {
  return cross(a,b,c).dot(p-a); }
bool coplanar(const P3&a,const P3&b,const P3&c,const P3&p) {
  return DC(a,b,c,p) == 0; }
bool above(const P3&a,const P3&b,const P3&c,const P3&p) {
  return DC(a,b,c,p) > 0; } // is p strictly above plane
void prep(vector<P3>& p) { // rearrange points such that
  shuffle(all(p),rng); // first four are not coplanar
  int dim = 1;
  rep(i,1,sz(p))
    if (dim == 1) {
      if (p[0] != p[i]) swap(p[1],p[i]), ++dim;
    } else if (dim == 2) {
      if (!collinear(p[0],p[1],p[i]))
        swap(p[2],p[i]), ++dim;
    } else if (dim == 3) {
      if (!coplanar(p[0],p[1],p[2],p[i]))
        swap(p[3],p[i]), ++dim;
    }
  assert(dim == 4);
}
```

3dHull-slow-benq.h
Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.
Time: $\mathcal{O}\left(n^2\right)$
0ba821, 27 lines

```cpp
using F = array<int,3>; // face
vector<F> hull3d(vector<P3>& p) {
  // s.t. first four points form tetra
  prep(p); int N = sz(p); vector<F> hull; // triangle for
      each face
  auto ad = [&](int a, int b, int c) { hull.pb({a,b,c}); };
```

```cpp
  // +new face to hull
  ad(0,1,2), ad(0,2,1); // initialize hull as first 3 points
  vector<vb> in(N,vb(N)); // is zero before each iteration
  rep(i,3,N) { // incremental construction
    vector<F> def, HULL; swap(hull,HULL);
    // HULL now contains old hull
    auto ins = [&](int a, int b, int c) {
      if (in[b][a]) in[b][a] = 0; // kill reverse face
      else in[a][b] = 1, ad(a,b,c);
    };
    for (auto &f:HULL) {
      if (above(p[f[0]],p[f[1]],p[f[2]],p[i]))
        rep(j,0,3) ins(f[j],f[(j+1)%3],i);
        // recalc all faces s.t. point is above face
      else def.pb(f);
    }
    for (auto &t:hull) if (in[t[0]][t[1]]) // edge exposed,
      in[t[0]][t[1]] = 0, def.pb(t); // add a new face
    swap(hull,def);
  }
  return hull;
}
```

3dHull-fast-benq.h
Description: Computes all faces of the 3-dimension hull of a point set.
Time: $\mathcal{O}\left(n \log n\right)$
c9ba16, 46 lines

```cpp
using F = array<int,3>; // face
vector<F> hull3dFast(vector<P3>& p) {
  prep(p); int N = sz(p); vector<F> hull;
  vb active; // whether face is active
  vector<vi> rvis; // points visible from each face
  vector<array<pi,3>> other; // other face adjacent to each
      edge of face
  vector<vi> vis(N); // faces visible from each point
  auto ad = [&](int a, int b, int c) {
    hull.pb({a,b,c}); active.pb(1); rvis.emplace_back();
    other.emplace_back(); };
  auto ae = [&](int a, int b) { vis[b].pb(a), rvis[a].pb(b);
      };
  auto abv = [&](int a, int b) {
    F f=hull[a]; return above(p[f[0]],p[f[1]],p[f[2]],p[b])
      ;};
  auto edge = [&](pi e) -> pi {
    return {hull[e.first][e.second],hull[e.first][(e.second+
      1)%3]}; };
  auto glue = [&](pi a, pi b) { // link two faces by an edge
    pi x = edge(a); assert(edge(b) == mp(x.second,x.first));
    other[a.first][a.second] = b, other[b.first][b.second] =
      a;
  }; // ensure face 0 is removed when i=3
  ad(0,1,2), ad(0,2,1); if (abv(1,3)) swap(p[1],p[2]);
  rep(i,0,3) glue({0,i},{1,2-i});
  rep(i,3,N) ae(abv(1,i),i); // coplanar points go in rvis
      [0]
  vi label(N,-1);
  rep(i,3,N) { // incremental construction
    vi rem; for(auto &t:vis[i]) if (active[t]) active[t]=0,
      rem.pb(t);
    if (!sz(rem)) continue; // hull unchanged
    int st = -1;
    for(auto &r:rem) rep(j,0,3) {
      int o = other[r][j].first;
      if (active[o]) { // create new face!
        int a,b; tie(a,b) = edge({r,j}); ad(a,b,i); st = a;
        int cur = sz(rvis)-1; label[a] = cur;
        vi tmp; set_union(all(rvis[r]),all(rvis[o]),
            back_inserter(tmp));
        // merge sorted vectors ignoring duplicates
        for(auto &x:tmp) if (abv(cur,x)) ae(cur,x);
        glue({cur,0},other[r][j]); // glue old w/ new face
      }
    }
    for (int x = st, y; ; x = y) { // glue new faces
      together
      int X = label[x]; glue({X,1},{label[y=hull[X][1]],2});
      if (y == st) break;
    }
  }
  vector<F> ans; rep(i,0,sz(hull)) if (active[i]) ans.pb(
    hull[i]);
  return ans;
}
```

sphericalDistance.h
Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.
611f07, 8 lines

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

Strings (9)

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: $\mathcal{O}\left(n\right)$
291ecf, 26 lines

```cpp
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;
}
vi match2(const string& s, const string& pat) { // only
    compute pi for pat
  vi p = pi(pat), res;
  int cp = 0;
  rep(i,1,sz(s)) {
    int g = cp;
    while (g && s[i] != pat[g]) g = p[g-1];
    cp = g + (s[i] == pat[g]);
    if (cp >= sz(pat)) res.push_back(i - sz(pat) + 1);
  }
  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: $\mathcal{O}\left(n\right)$
ee09e2, 12 lines

```cpp
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 (i is right of middle, s[i]==s[i-1]), p[1][i] = longest odd (half rounded down).
Time: $\mathcal{O}\left(N\right)$
e7ad79, 13 lines

```cpp
array<vi, 2> manacher(const string& s) {
  int n = sz(s);
```

```
3    array<vi,2> p = {vi(n+1), vi(n)};
4    rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
5      int t = r-i+!z;
6      if (i<r) p[z][i] = min(t, p[z][l+t]);
7      int L = i-p[z][i], R = i+p[z][i]-!z;
8      while (L>=1 && R+1<n && s[L-1] == s[R+1])
9        p[z][i]++, L--, R++;
10     if (R>r) l=L, r=R;
11   }
12   return p;
13 }
```

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

```
1  int minRotation(string s) {
2    int a=0, N=sz(s); s += s;
3    rep(b,0,N) rep(k,0,N) {
4      if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1);
5        break;}
6      if (s[a+k] > s[b+k]) { a = b; break; }
7    }
8    return a;
9  }
```

SuffixArray.h
Description: Builds suffix array for a string. sa[i] is the starting index of the suffix which is i'th in the sorted suffix array. The returned vector is of size $n + 1$, and sa[0] = n. The lcp array contains longest common prefixes for neighbouring strings in the suffix array: lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0. The input string must not contain any zero bytes.
Time: $\mathcal{O}(n \log n)$ ef440a, 44 lines

```
1  struct SuffixArray {
2    vi sa, lcp; // sa[0] is empty str, size is n+1, lcp[i] is
      of sa[i] and sa[i-1]
3    SuffixArray(string& s, int lim=256) { // or basic_string<
      int>
4      int n = sz(s) + 1, k = 0, a, b;
5      vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
6      sa = lcp = y, iota(all(sa), 0);
7      for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p
        ) {
8        p = j, iota(all(y), n - j);
9        rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
10       fill(all(ws), 0);
11       rep(i,0,n) ws[x[i]]++;
12       rep(i,1,lim) ws[i] += ws[i - 1];
13       for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
14       swap(x, y), p = 1, x[sa[0]] = 0;
15       rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
16         (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p
         ++;
17     }
18     rep(i,1,n) rank[sa[i]] = i;
19     for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
20       for (k && k--, j = sa[rank[i] - 1];
21         s[i + k] == s[j + k]; k++);
22   }
23   // total unique substrings = (n+1 C 2) - sum(lcp)
24   // string search in O(m log n). returns [l,r] of matches
      in sa.
25   pair<int,int> search(string &t) {
26     int b=1,e=sz(sa)-1;
27     while (b<e) {
28       int m=(b+e)/2;
29       if (s.compare(sa[m], sz(t), t) >= 0) e = m; // not
         strict
30       else b = m+1;
31     }
32     if (s.compare(sa[b], sz(t), t) != 0) return mp(-1,-1);
33     pair<int,int> ans{b,-1};
34     b=1,e=sz(sa)-1;
35     while (b<e) {
36       int m=(b+e)/2;
37       if (s.compare(sa[m], sz(t), t) > 0) e = m; // strict
38       else b = m+1;
39     }
40     if (s.compare(sa[b], sz(t), t) != 0) e--;
41     ans.second = e;
42     return ans;
```

```
43   }
44 };
```

SuffixAutomaton.h
Description: what it says
Usage: just do it   bfbd86, 39 lines

```
1  struct SA {
2    vector<map<char, int> > adj;
3    vi link, dis;
4    int N;
5    SA(): adj(1), link(1, -1), dis(1, 0), N(1) {}
6    int new_node(int v=-1) {
7      if(v == -1)
8        adj.emplace_back(), link.emplace_back(), dis.
         emplace_back();
9      else
10       adj.push_back(adj[v]), link.push_back(link[v]), dis.
         push_back(dis[v]);
11     return N++;
12   }
13   int go(int p, int c) {
14     auto it = adj[p].find(c);
15     if(dis[it->second] == dis[p] + 1)
16       return it->second;
17     else {
18       int q = it->second, n = new_node(q);
19       dis[n] = dis[p] + 1, link[q] = n;
20       for(;p != -1 && (it = adj[p].find(c))->second == q;p =
         link[p])
21         it->second = n;
22       return n;
23     }
24   }
25   int append(int p, char c) {
26     auto it = adj[p].find(c);
27     if(it != adj[p].end())
28       return go(p, c);
29     int n = new_node();
30     dis[n] = dis[p] + 1;
31     for(;p != -1 && adj[p].find(c) == adj[p].end();p = link[
         p])
32       adj[p].insert({c, n});
33     if(p == -1)
34       link[n] = 0;
35     else
36       link[n] = go(p, c);
37     return n;
38   }
39 };
```

Hashing.h
Description: Self-explanatory methods for string hashing.   2d2a67, 44 lines

```
1  // Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
2  // code, but works on evil test data (e.g. Thue-Morse, where
3  // ABBA... and BAAB... of length 2^10 hash the same mod
     2^64).
4  // "typedef ull H;" instead if you think test data is random
     ,
5  // or work mod 10^9+7 if the Birthday paradox is not a
     problem.
6  typedef uint64_t ull;
7  struct H {
8    ull x; H(ull x=0) : x(x) {}
9    H operator+(H o) { return x + o.x + (x + o.x < x); }
10   H operator-(H o) { return *this + ~o.x; }
11   H operator*(H o) { auto m = (__uint128_t)x * o.x;
12     return H((ull)m) + (ull)(m >> 64); }
13   ull get() const { return x + !~x; }
14   bool operator==(H o) const { return get() == o.get(); }
15   bool operator<(H o) const { return get() < o.get(); }
16 };
17 static const H C = (ll)1e11+3; // (order ~ 3e9; random also
     ok)
18
19 struct HashInterval {
20   vector<H> ha, pw;
21   HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
22     pw[0] = 1;
23     rep(i,0,sz(str))
24       ha[i+1] = ha[i] * C + str[i],
25       pw[i+1] = pw[i] * C;
```

```
26   }
27   H hashInterval(int a, int b) { // hash [a, b)
28     return ha[b] - ha[a] * pw[b - a];
29   }
30 };
31
32 vector<H> getHashes(string& str, int length) {
33   if (sz(str) < length) return {};
34   H h = 0, pw = 1;
35   rep(i,0,length)
36     h = h * C + str[i], pw = pw * C;
37   vector<H> ret = {h};
38   rep(i,length,sz(str)) {
39     ret.push_back(h = h * C + str[i] - pw * str[i-length]);
40   }
41   return ret;
42 }
43
44 H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return
     h;}
```

PalTree.h
Description: trie of right-half of all palindromic substrings link points to longest suffix-palindrome of node oc stores num occurences after resolveOc() ans[i][b] is min factorization of prefix i (1-indexed) with parity b
Time: $\mathcal{O}(n \log n)$ 0a57d3, 39 lines

```
1  struct PalTree {
2    static const int ASZ = 26;
3    struct node {
4      array<int,ASZ> to = array<int,ASZ>();
5      int len, link, oc = 0; // # occurrences of pal
6      int slink = 0, diff = 0;
7      array<int,2> seriesAns;
8      node(int _len, int _link) : len(_len), link(_link)
       {}
9    };
10   string s = "@"; vector<array<int,2>> ans = {{0,INF}};
11   vector<node> d = {{0,1},{-1,0}}; // dummy pals of len
      0,-1
12   int last = 1;
13   int getLink(int v) {
14     while (s[sz(s)-d[v].len-2] != s.back()) v = d[v].
        link;
15     return v;
16   }
17   void updAns() { // serial path has O(log n) vertices
18     ans.pb({INF,INF});
19     for (int v = last; d[v].len > 0; v = d[v].slink) {
20       d[v].seriesAns=ans[sz(s)-1-d[d[v].slink].len-d[v
        ].diff];
21       if (d[v].diff == d[d[v].link].diff)
22         rep(i,0,2) ckmin(d[v].seriesAns[i], d[d[v].
        link].seriesAns[i]);
23       // start of previous oc of link[v]=start of last
        oc of v
24       rep(i,0,2) ckmin(ans.back()[i],d[v].seriesAns[i^
        1]+1);
25     }
26   }
27   void addChar(char C) {
28     s += C; int c = C-'a'; last = getLink(last);
29     if (!d[last].to[c]) {
30       d.emplace_back(d[last].len+2,d[getLink(d[last].
        link)].to[c]);
31       d[last].to[c] = sz(d)-1;
32       auto& z = d.back(); z.diff = z.len-d[z.link].len
        ;
33       z.slink = z.diff == d[z.link].diff ? d[z.link].
        slink : z.link;
34     } // max suf with different dif
35     last = d[last].to[c]; ++d[last].oc;
36     updAns();
37   }
38   void resolveOc() { for (int i=sz(d)-1;i>=2;i--) d[d[i].
      link].oc += d[i].oc; }
39 };
```

AhoCorasickJacob.h
Description: Aho-Corasick with unfixed, trie of all strings Link stores node of longest suffix that is also in the trie Only works after adding all strings and calling build().   5cfbad, 44 lines

```
1  struct AhoCorasick {
2      struct node {
3          unordered_map<char,int> next; // can use array
4          int link = -1;
5          int output = 0; // adapt as necessary
6          node() {}
7      };
8      vector<node> nodes{1};
9      void add_string(const string& s) {
10         int v = 0;
11         for (char c:s) {
12             if (nodes[v].next.find(c) == nodes[v].next.end()
    ) {
13                 nodes[v].next[c] = sz(nodes);
14                 nodes.emplace_back();
15             }
16             v = nodes[v].next[c];
17         }
18         nodes[v].output = 1; // update output
19     }
20     int nx(int j, char c) {
21         while (j != -1 && !nodes[j].next.count(c)) j = nodes
    [j].link;
22         return (j == -1) ? 0 : nodes[j].next[c];
23     }
24     void build() {
25         queue<int> q; q.push(0);
26         while (!q.empty()) {
27             int v = q.front(); q.pop();
28             for (auto p : nodes[v].next) {
29                 int x = p.second;
30                 nodes[x].link = nx(nodes[v].link, p.first);
31                 // update output if needed
32                 q.push(x);
33             }
34         }
35     }
36     int match(const string &text) {
37         int n = sz(text), v = 0;
38         for (int i = 0; i < n; i++) {
39             v = nx(v, text[i]);
40             if (nodes[v].output) return i;
41         }
42         return -1;
43     }
44 };
```

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

```
1  set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
2      if (L == R) return is.end();
3      auto it = is.lower_bound({L, R}), before = it;
4      while (it != is.end() && it->first <= R) {
5          R = max(R, it->second);
6          before = it = is.erase(it);
7      }
8      if (it != is.begin() && (--it)->second >= L) {
9          L = min(L, it->first);
10         R = max(R, it->second);
11         is.erase(it);
12     }
13     return is.insert(before, {L,R});
14 }
15
16 void removeInterval(set<pii>& is, int L, int R) {
17     if (L == R) return;
18     auto it = addInterval(is, L, R);
19     auto r2 = it->second;
20     if (it->first == L) is.erase(it);
21     else (int&)it->second = L;
22     if (R != r2) is.emplace(R, r2);
23 }
```

IntervalCover.h

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

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

9e9d8d, 19 lines

```
1  template<class T>
2  vi cover(pair<T, T> G, vector<pair<T, T>> I) {
3      vi S(sz(I)), R;
4      iota(all(S), 0);
5      sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
6      T cur = G.first;
7      int at = 0;
8      while (cur < G.second) { // (A)
9          pair<T, int> mx = make_pair(cur, -1);
10         while (at < sz(I) && I[S[at]].first <= cur) {
11             mx = max(mx, make_pair(I[S[at]].second, S[at]));
12             at++;
13         }
14         if (mx.second == -1) return {};
15         cur = mx.first;
16         R.push_back(mx.second);
17     }
18     return R;
19 }
```

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

```
1  template<class F, class G, class T>
2  void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
3      if (p == q) return;
4      if (from == to) {
5          g(i, to, p);
6          i = to; p = q;
7      } else {
8          int mid = (from + to) >> 1;
9          rec(from, mid, f, g, i, p, f(mid));
10         rec(mid+1, to, f, g, i, p, q);
11     }
12 }
13 template<class F, class G>
14 void constantIntervals(int from, int to, F f, G g) {
15     if (to <= from) return;
16     int i = from; auto p = f(i), q = f(to-1);
17     rec(from, to-1, f, g, i, p, q);
18     g(i, to, q);
19 }
```

10.2 Misc. algorithms

Dates.h

Description: Dates

74f735, 42 lines

```
1  // Routines for performing computations on dates. In these
       routines,
2  // months are expressed as integers from 1 to 12, days are
       expressed
3  // as integers from 1 to 31, and years are expressed as 4-
       digit
4  // integers.
5  string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "
       Sat", "Sun"};
6  // converts Gregorian date to integer (Julian day number)
7  int dateToInt (int m, int d, int y){
8      return
9          1461 * (y + 4800 + (m - 14) / 12) / 4 +
10         367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
11         3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
12         d - 32075;
13 }
14 // converts integer (Julian day number) to Gregorian date:
       month/day/year
15 void intToDate (int jd, int &m, int &d, int &y){
16     int x, n, i, j;
17     x = jd + 68569;
18     n = 4 * x / 146097;
```

```
19     x -= (146097 * n + 3) / 4;
20     i = (4000 * (x + 1)) / 1461001;
21     x -= 1461 * i / 4 - 31;
22     j = 80 * x / 2447;
23     d = x - 2447 * j / 80;
24     x = j / 11;
25     m = j + 2 - 12 * x;
26     y = 100 * (n - 49) + i + x;
27 }
28 // converts integer (Julian day number) to day of week
29 string intToDay (int jd){ return dayOfWeek[jd % 7]; }
30 /*int main (int argc, char **argv){
31     int jd = dateToInt (3, 24, 2004);
32     int m, d, y;
33     intToDate (jd, m, d, y);
34     string day = intToDay (jd);
35     // expected output:
36     //    2453089
37     //    3/24/2004
38     //    Wed
39     cout << jd << endl
40         << m << "/" << d << "/" << y << endl
41         << day << endl;
42 }*/
```

TernarySearch.h

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

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

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

9155b4, 11 lines

```
1  template<class F>
2  int ternSearch(int a, int b, F f) {
3      assert(a <= b);
4      while (b - a >= 5) {
5          int mid = (a + b) / 2;
6          if (f(mid) < f(mid+1)) a = mid; // (A)
7          else b = mid+1;
8      }
9      rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
10     return a;
11 }
```

LIS.h

Description: Compute indices for the longest increasing subsequence.

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

2932a0, 17 lines

```
1  template<class I> vi lis(const vector<I>& S) {
2      if (S.empty()) return {};
3      vi prev(sz(S));
4      typedef pair<I, int> p;
5      vector<p> res;
6      rep(i,0,sz(S)) {
7          // change 0 -> i for longest non-decreasing subsequence
8          auto it = lower_bound(all(res), p{S[i], 0});
9          if (it == res.end()) res.emplace_back(), it = res.end()-
    1;
10         *it = {S[i], i};
11         prev[i] = it == res.begin() ? 0 : (it-1)->second;
12     }
13     int L = sz(res), cur = res.back().second;
14     vi ans(L);
15     while (L--) ans[L] = cur, cur = prev[cur];
16     return ans;
17 }
```

FastKnapsack.h

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

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

b20ccc, 16 lines

```
1  int knapsack(vi w, int t) {
2      int a = 0, b = 0, x;
3      while (b < sz(w) && a + w[b] <= t) a += w[b++];
4      if (b == sz(w)) return a;
5      int m = *max_element(all(w));
6      vi u, v(2*m, -1);
7      v[a+m-t] = b;
8      rep(i,b,sz(w)) {
9          u = v;
```

```
10    rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
11    for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
12      v[x-w[j]] = max(v[x-w[j]], j);
13  }
14  for (a = t; v[a+m-t] < 0; a--) ;
15  return a;
16 }
```

10.3 Dynamic programming

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

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: $\mathcal{O}((N + (hi - lo))\log N)$ d38d2b, 18 lines

```
1  struct DP { // Modify at will:
2    int lo(int ind) { return 0; }
3    int hi(int ind) { return ind; }
4    ll f(int ind, int k) { return dp[ind][k]; }
5    void store(int ind, int k, ll v) { res[ind] = pii(k, v); }
6
7    void rec(int L, int R, int LO, int HI) {
8      if (L >= R) return;
9      int mid = (L + R) >> 1;
10     pair<ll, int> best(LLONG_MAX, LO);
11     rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
12       best = min(best, make_pair(f(mid, k), k));
13     store(mid, best.second, best.first);
14     rec(L, mid, LO, best.second+1);
15     rec(mid+1, R, best.second, HI);
16   }
17   void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
18 };
```

10.4 Debugging tricks

- signal(SIGSEGV, [](int) { _Exit(0); }); converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). _GLIBCXX_DEBUG failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).

- feenableexcept(29); kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

10.5 Optimization tricks

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

10.5.1 Bit hacks

- x & -x is the least bit in x.

- for (int x = m; x;) { --x &= m; ... } loops over all subset masks of m (except m itself).

- c = x&-x, r = x+c; (((r^x) >> 2)/c) | r is the next number after x with the same number of bits set.

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

10.5.2 Pragmas

- **#pragma** GCC optimize ("Ofast") will make GCC auto-vectorize loops and optimizes floating points better.

- **#pragma** GCC target ("avx2") can double performance of vectorized code, but causes crashes on old machines.

- **#pragma** GCC optimize ("trapv") kills the program on integer overflows (but is really slow).

FastMod.h
Description: Compute $a\%b$ about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to $a \pmod{b}$ in the range $[0, 2b)$.   751a02, 8 lines

```
1  typedef unsigned long long ull;
2  struct FastMod {
3    ull b, m;
4    FastMod(ull b) : b(b), m(-1ULL / b) {}
5    ull reduce(ull a) { // a % b + (0 or b)
6      return a - (ull)((__uint128_t(m) * a) >> 64) * b;
7    }
8  };
```

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

```
1  inline char gc() { // like getchar()
2    static char buf[1 << 16];
3    static size_t bc, be;
4    if (bc >= be) {
5      buf[0] = 0, bc = 0;
6      be = fread(buf, 1, sizeof(buf), stdin);
7    }
8    return buf[bc++]; // returns 0 on EOF
9  }
10
11 int readInt() {
12   int a, c;
13   while ((a = gc()) < 40);
14   if (a == '-') return -readInt();
15   while ((c = gc()) >= 48) a = a * 10 + c - 480;
16   return a - 48;
17 }
```

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

```
1  // Either globally or in a single class:
2  static char buf[450 << 20];
3  void* operator new(size_t s) {
4    static size_t i = sizeof buf;
5    assert(s < i);
6    return (void*)&buf[i -= s];
7  }
8  void operator delete(void*) {}
```

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

```
1  template<class T> struct ptr {
2    unsigned ind;
3    ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
4      assert(ind < sizeof buf);
5    }
6    T& operator*() const { return *(T*)(buf + ind); }
7    T* operator->() const { return &**this; }
8    T& operator[](int a) const { return (&**this)[a]; }
9    explicit operator bool() const { return ind; }
10 };
```