Indian Institute of Technology Kanpur

bytes_please

Umang Malik, Aryan Choudhary, Abhay Pratap Singh

2019-12-25

Contents

Contest (1)

template.cpp — 26 lines

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i, a, b) for(int i = a; i < (b);
    ++i)
#define trav(a, x) for(auto& a : x)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;//lli
typedef pair<int, int> pii;
typedef vector<int> vi;// long long
mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
//{return uniform_int_distribution<lli>(l,r)
    (rng);}
#define dbg(...) { cerr<<"[ "; ary(#
    __VA_ARGS__, __VA_ARGS__);}
template <typename Arg1>
void ary(const string name, Arg1&& arg1){
    cerr << name << " : " << arg1 << " ] "
    << endl;}
template <typename Arg1, typename... Args>
void ary(const string names, Arg1&& arg1,
    Args&&... args){
  const string name = names.substr(0,names.
      find(',');cerr<<name<<" : "<<arg1<<"
      | ";
  ary(names.substr(1+(int)name.size()), args
      ...);
}
template <typename Arg1,typename Arg2>
ostream& operator << (ostream& out, const
    map<Arg1,Arg2> &a) {
  out<<"[";for(const auto &x:a)out<<x<<",";
      return out<<"]";
}
int main() {
  cin.sync_with_stdio(0); cin.tie(0);cin.
      exceptions(cin.failbit);
}
```

.bashrc — 5 lines

```
#!/bin/zsh
file=${1:-a.cpp}
g++ -D ARYANC403=1 ${file} -o ${file}.out -
    fsanitize=address,undefined,signed-
    integer-overflow -Wall -std=gnu++17 &&
    time ./${file}.out ${2}
g++ ${file} -o ${file}.out -std=gnu++14 &&
    time ./${file}.out ${2}
cpp -dD -P -fpreprocessed | tr -d '[:space:]
    '| md5sum |cut -c-6 //hash.sh
```

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

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

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

2.3 Trigonometry and Geometry

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

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

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

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

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

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

2.3.1 Triangles

Side lengths: a, b, c

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

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

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

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

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

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

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

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

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

2.3.2 Quadrilaterals

With side lengths a, b, c, d, diagonals e, f, diagonals angle 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 $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.3.3 Spherical coordinates

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

2.4 Derivatives/Integrals

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

Integration by parts:

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

2.5 Sums and Series

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

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

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

2.6 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y,

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

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 $\text{Po}(\lambda)$, $\lambda = t\kappa$.

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

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

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

2.7 Markov chains

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

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

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

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

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 $t_i = 1 + \sum_{k\in\mathbf{G}} p_{ki}t_k$.

Data structures (3)

OrderStatisticTree.h

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

Time: $\mathcal{O}(\log N)$ 782797, 16 lines

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

HashMap.h

Description: Hash map with the same API as unordered_map, but ~3x faster. Initial capacity must be a power of 2 (if provided). 1443bc, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the
    lowest ones:
struct chash {
  const uint64_t C = ll(2e18 * M_PI) + 71;
      // large odd number
  ll operator()(ll x) const { return
      __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h
    ({},{},{},{},{1<<16});
```

SegmentTree.h

Description: Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying T, f and unit.

Time: $\mathcal{O}(\log N)$ 0f4bdb, 19 lines

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

LazySegmentTree.h

Description: Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.

Usage: `Node* tr = new Node(v, 0, sz(v));`

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

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

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

UnionFind.h

Description: Disjoint-set data structure.

Time: $\mathcal{O}(\alpha(N))$ b5bfc3, 14 lines

```
struct UF {
  vi e;
  UF(int n) : e(n, -1) {}
  bool same_set(int a, int b) { return find(
      a) == find(b); }
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : e[
      x] = find(e[x]); }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

SubMatrix.h

Description: Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).

Usage: `SubMatrix<int> m(matrix);`
`m.sum(0, 0, 2, 2); // top left 4 elements`

Time: $\mathcal{O}(N^2 + Q)$ c59ada, 13 lines

```
template<class T>
struct SubMatrix {
  vector<vector<T>> p;
  SubMatrix(vector<vector<T>>& v) {
    int R = sz(v), C = sz(v[0]);
    p.assign(R+1, vector<T>(C+1));
    rep(r,0,R) rep(c,0,C)
      p[r+1][c+1] = v[r][c] + p[r][c+1] + p[
          r+1][c] - p[r][c];
  }
  T sum(int u, int l, int d, int r) {
    return p[d][r] - p[d][l] - p[u][r] + p[u
        ][l];
  }
};
```

Matrix.h

Description: Basic operations on square matrices.

Usage: `Matrix<int, 3> A;`
`A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};`
`vector<int> vec = {1,2,3};`
`vec = (A^N) * vec;` c43c7d, 26 lines

```
template<class T, int N> struct Matrix {
  typedef Matrix M;
  array<array<T, N>, N> d{};
  M operator*(const M& m) const {
    M a;
    rep(i,0,N) rep(j,0,N)
      rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k
          ][j];
    return a;
  }
  vector<T> operator*(const vector<T>& vec)
      const {
    vector<T> ret(N);
    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j]
        * vec[j];
    return ret;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      p >>= 1;
    }
    return a;
  }
};
```

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.

Time: $\mathcal{O}(\log N)$ 95e223, 30 lines

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

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}(\log N)$ 9556fc, 55 lines

```
struct Node {
  Node *l = 0, *r = 0;
  int val, y, c = 1;
  Node(int val) : val(val), y(rand()) {}
  void recalc();
};
int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) +
    1; }
template<class F> void each(Node* n, F f) {
  if (n) { each(n->l, f); f(n->val); each(n
      ->r, f); }
}
pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  if (cnt(n->l) >= k) { // "n->val >= k" for
      lower_bound(k)
    auto pa = split(n->l, k);
    n->l = pa.second;
    n->recalc();
    return {pa.first, n};
  } else {
    auto pa = split(n->r, k - cnt(n->l) - 1)
        ; // and just "k"
    n->r = pa.first;
    n->recalc();
    return {n, pa.second};
  }
}
Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  if (l->y > r->y) {
    l->r = merge(l->r, r);
    l->recalc();
    return l;
  } else {
    r->l = merge(l, r->l);
    r->recalc();
    return r;
  }
}
Node* ins(Node* t, Node* n, int pos) {
  auto pa = split(t, pos);
  return merge(merge(pa.first, n), pa.second
      );
}
// Example application: move the range [l, r
    ) to index k
void move(Node*& t, int l, int r, int k) {
  Node *a, *b, *c;
  tie(a,b) = split(t, l); tie(b,c) = split(b
      , r - l);
  if (k <= l) t = merge(ins(a, b, k), c);
  else t = merge(a, ins(c, b, k - r));
}
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
Time: Both operations are $\mathcal{O}(\log N)$. e62fac, 22 lines

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

FenwickTree2d.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).
Time: $\mathcal{O}(\log^2 N)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)
"FenwickTree.h" b28c27, 22 lines

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

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage: `RMQ rmq(values);`
`rmq.query(inclusive, exclusive);`
Time: $\mathcal{O}(|V|\log|V| + Q)$ 1f8996, 17 lines

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

Numerical (4)

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: $\mathcal{O}(\log((b-a)/\epsilon))$ 31d45b, 14 lines

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

Polynomial.h

c9b7b0, 17 lines

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

PolyRoots.h

Description: Finds the real roots to a polynomial.

Usage: `poly_roots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`

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

"Polynomial.h" 2cf190, 23 lines

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

PolyInv.h

Description: Find inverse of polynomial h(x) mod x^{ll}.

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

705583, 22 lines

```
vl inv(vl& h,int ll){
    assert((ll&(ll-1))==0);
    if(sz(h)<ll) h.resize(ll,0);
    vl ans;
    ans.pb(modpow(h[0],mod-2));
    for(int l = 2;l<=ll;l*=2){
        vl a = ans;
        vl h0 = vl(h.begin(),h.begin()+l/2);
        vl h1 = vl(h.begin()+l/2,h.begin()+l);
        vl c = conv(a,h0);
        if(sz(c)<l/2+1) c.resize(l/2+1,0);
        c = vl(c.begin()+l/2,c.end());
        vl tem = add(c,conv(a,h1));
        tem.resize(l/2,0);
        vl b = conv(a,tem);
        b.resize(l/2,0);
        trav(i,b) ans.pb(-i);
    }
    // while(sz(ans)>1&&ans.back()==0) ans.pop_back();
    while(sz(h)>1&&h.back()==0) h.pop_back();
    return ans;
}
```

PolyInterpolate.h

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

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

08bf48, 13 lines

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

BerlekampMassey.h

Description: Recovers any n-order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$.

Usage: `BerlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}`

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

"../number-theory/ModPow.h" 40387d, 20 lines

```
vector<ll> BerlekampMassey(vector<ll> s) {
  int n = sz(s), L = 0, m = 0;
  vector<ll> C(n), B(n), T;
  C[0] = B[0] = 1;
  ll b = 1;
  rep(i,0,n) { ++m;
    ll d = s[i] % mod;
    rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
    if (!d) continue;
    T = C; ll coef = d * modpow(b, mod-2) % mod;
    rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
    if (2 * L > i) continue;
    L = i + 1 - L; B = T; b = d; m = 0;
  }
  C.resize(L + 1); C.erase(C.begin());
  trav(x, C) x = (mod - x) % mod;
  return C;
}
```

LinearRecurrence.h

Description: Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \dots \geq n-1]$ and $tr[0 \dots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.

Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`

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

f4e444, 26 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(tr);
  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    rep(i,0,n+1) rep(j,0,n+1)
      res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i) rep(j,0,n)
      res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
    res.resize(n + 1);
    return res;
  };
  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;
  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }
  ll res = 0;
  rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

HillClimbing.h

Description: Poor man's optimization for unimodal functions.

f40e55, 16 lines

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

Integrate.h

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

4756fc, 7 lines

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

IntegrateAdaptive.h

Description: Fast integration using an adaptive Simpson's rule.

Usage:
```
double sphereVolume = quad(-1, 1, [](double x) {
return quad(-1, 1, [&](double y) {
return quad(-1, 1, [&](double z) {
return x*x + y*y + z*z < 1; });});});
```

92dd79, 15 lines

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

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

Determinant.h

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

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

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}\left(N^3\right)$ 3313dc, 18 lines

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

Simplex.h

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

```
typedef double T; // long double, Rational,
    double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;
const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) <
     MP(X[s],N[s])) s=j
struct LPSolver {
  int m, n;
  vi N, B;
  vvd D;
  LPSolver(const vvd& A, const vd& b, const
      vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2,
        vd(n+2)) {
      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j
          ];
      rep(i,0,m) { B[i] = n+i; D[i][n] = -1;
           D[i][n+1] = b[i];}
      rep(j,0,n) { N[j] = j; D[m][j] = -c[j
          ]; }
      N[n] = -1; D[m+1][n] = 1;
    }
  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s])
        > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv
        ;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }
  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D
          [x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s
            ], B[i])
                     < MP(D[r][n+1] / D[r][s
                         ], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }
  T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r
        = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps)
        return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][
        n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

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: $\mathcal{O}\left(n^2 m\right)$ 44c9ab, 38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;
int solveLinear(vector<vd>& A, vd& b, vd& x)
     {
  int n = sz(A), m = sz(x), rank = 0, br, bc
      ;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps)
        return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }
  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if
      rank < m)
}
```

SolveLinear2.h

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

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

SolveLinearBinary.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}(n^2 m)$ fa2d7a, 34 lines

```
typedef bitset<1000> bs;
int solveLinear(vector<bs>& A, vi& b, bs& x,
   int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any())
      break;
    if (br == n) {
      rep(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);
    }
    rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
    }
    rank++;
  }
  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if
     rank < m)
}
```

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}(n^3)$ ebfff6, 35 lines

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

Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

4.1 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. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs). Otherwise, use long doubles/NTT/FFTMod.
Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$) c38f45, 35 lines

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2,
     1);
  static vector<C> rt(2, 1);  // (^ 10%
     faster if double)
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, M_PIl / k); // M_PI
      , lower-case L
    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2]
        * x : R[i/2];
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1)
     << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[
     rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j
      ,0,k) {
      C z = rt[j+k] * a[i+j+k]; (25% faster
        if hand-rolled)
      a[i + j + k] = a[i + j] - z;
      a[i + j] += z;
    }
}
vd conv(const vd& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1
     << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i,0,sz(b)) in[i].imag(b[i]);
  fft(in);
  trav(x, in) x *= x;
  rep(i,0,n) out[i] = in[-i & (n - 1)] -
     conj(in[i]);
  fft(out);
  rep(i,0,sz(res)) res[i] = imag(out[i]) /
     (4 * n);
  return res;
}
```

FastFourierTransformMod.h

Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.
Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)
"FastFourierTransform.h" b8e1bf, 22 lines

```
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a,
   const vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B=32-__builtin_clz(sz(res)), n=1<<B,
     cut=int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (
      int)a[i] % cut);
  rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (
      int)b[i] % cut);
  fft(L), fft(R);
  rep(i,0,n) {
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] /
        (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] /
        (2.0 * n) / C(1.0i);
  }
  fft(outl), fft(outs);
  rep(i,0,sz(res)) {
    ll av = ll(real(outl[i])+.5), cv = ll(
        imag(outs[i])+.5);
    ll bv = ll(imag(outl[i])+.5) + ll(real(
        outs[i])+.5);
    res[i] = ((av % M * cut + bv) % M * cut
        + cv) % M;
  }
  return res;
}
```

NumberTheoreticTransform.h
Description: Can be used for convolutions modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. Inputs must be in [0, mod).
Time: $\mathcal{O}(N \log N)$
"../number-theory/ModPow.h" d75aad, 32 lines

```
const ll mod = (119 << 23) + 1, root = 62;
    // = 998244353
// For p < 2^30 there is also e.g. 5 << 25,
    7 << 26, 479 << 21
// and 483 << 21 (same root). The last two
    are > 10^9.
typedef vector<ll> vl;
void ntt(vl& a, vl& rt, vl& rev, int n) {
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[
      rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j
        ,0,k) {
        ll z = rt[j + k] * a[i + j + k] %
            mod, &ai = a[i + j];
        a[i + j + k] = (z > ai ? ai - z +
            mod : ai - z);
        ai += (ai + z >= mod ? z - mod : z);
    }
}
vl conv(const vl& a, const vl& b) {
  if (a.empty() || b.empty())
    return {};
  int s = sz(a)+sz(b)-1, B = 32 -
      __builtin_clz(s), n = 1 << B;
  vl L(a), R(b), out(n), rt(n, 1), rev(n);
  L.resize(n), R.resize(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1)
      << B) / 2;
  ll curL = mod / 2, inv = modpow(n, mod -
      2);
  for (int k = 2; k < n; k *= 2) {
    ll z[] = {1, modpow(root, curL /= 2)};
    rep(i,k,2*k) rt[i] = rt[i / 2] * z[i &
        1] % mod;
  }
  ntt(L, rt, rev, n); ntt(R, rt, rev, n);
  rep(i,0,n) out[-i & (n-1)] = L[i] * R[i] %
      mod * inv % mod;
  ntt(out, rt, rev, n);
  return {out.begin(), out.begin() + s};
}
```

FastSubsetTransform.h
Description: Transform to a basis with fast convolutions of the form $c[z] = \sum_{z=x\oplus y} a[x] \cdot b[y]$, where $\oplus$ is one of AND, OR, XOR. The size of a must be a power of two.
Time: $\mathcal{O}(N \log N)$ 3de473, 16 lines

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

4.2 Convolutions

AndConvolve.h
Description: It transform P for and conv. Assumes P has 2^n coeff. 27323f, 18 lines

```
vll transform(vll P, bool inverse) {
    for (len = 1; 2 * len <= degree(P); len
        <<= 1) {
        for (i = 0; i < degree(P); i += 2 *
            len) {
            for (j = 0; j < len; j++) {
                u = P[i + j];
                v = P[i + len + j];
                if (!inverse) {
                    P[i + j] = v;
                    P[i + len + j] = u + v;
                } else {
                    P[i + j] = -u + v;
                    P[i + len + j] = u;
                }
            }
        }
    }
    return P;
}
```

XorConvolve.h
Description: It transform P for xor conv. Assumes P has 2^n coeff. 5551fc, 17 lines

```
vll FWHT(vll P, bool inverse) {
    for (len = 1; 2 * len <= degree(P); len
        <<= 1) {
        for (i = 0; i < degree(P); i += 2 *
            len) {
            for (j = 0; j < len; j++) {
                u = P[i + j];
                v = P[i + len + j];
                P[i + j] = u + v;
                P[i + len + j] = u - v;
            }
        }
    }
    if (inverse) {
        for (i = 0; i < degree(P); i++)
            P[i] = P[i] / degree(P);
    }
    return P;
}
```

OrConvolve.h
Description: Matrix = [[1 1],[1 0]]

Number theory (5)

5.1 Modular arithmetic

ModularArithmetic.h
Description: Operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.
"euclid.h" 35bfea, 18 lines

```
const ll mod = 17; // change to something
    else
struct Mod {
  ll x;
  Mod(ll xx) : x(xx) {}
  Mod operator+(Mod b) { return Mod((x + b.x
      ) % mod); }
  Mod operator-(Mod b) { return Mod((x - b.x
      + mod) % mod); }
  Mod operator*(Mod b) { return Mod((x * b.x
      ) % mod); }
  Mod operator/(Mod b) { return *this *
      invert(b); }
  Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1); return Mod((x + mod) %
        mod);
  }
  Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2); r = r * r;
    return e&1 ? *this * r : r;
  }
};
```

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

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

ModPow.h b83e45, 8 lines

```
const ll mod = 1000000007; // faster if
    const
ll modpow(ll b, ll e) {
  ll ans = 1;
  for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
  return ans;
}
```

ModLog.h
Description: Returns the smallest $x \geq 0$ s.t. $a^x = b \pmod{m}$. a and m must be coprime.
Time: $\mathcal{O}(\sqrt{m})$ 49d606, 10 lines

```
ll modLog(ll a, ll b, ll m) {
  assert(__gcd(a, m) == 1);
  ll n = (ll) sqrt(m) + 1, e = 1, x = 1, res
       = LLONG_MAX;
  unordered_map<ll, ll> f;
  rep(i,0,n) e = e * a % m;
  rep(i,0,n) x = x * e % m, f.emplace(x, i +
       1);
  rep(i,0,n) if (f.count(b = b * a % m))
    res = min(res, f[b] * n - i - 1);
  return res;
}
```

ModSum.h

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

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

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

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \le a, b < c < 2^{63}$.
Time: $\mathcal{O}(1)$ for `mod_mul`, $\mathcal{O}(\log b)$ for `mod_pow` 88c37a, 12 lines

```
typedef unsigned long long ull;
typedef long double ld;
ull mod_mul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(ld(a) * ld(b) /
       ld(M));
  return ret + M * (ret < 0) - M * (ret >= (
       ll)M);
}
ull mod_pow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = mod_mul(b, b, mod), e /= 2)
    if (e & 1) ans = mod_mul(ans, b, mod);
  return ans;
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).
Time: $\mathcal{O}(\log^2 p)$ worst case, $\mathcal{O}(\log p)$ for most p
"ModPow.h" 19a793, 24 lines

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

Faulhaber.h

Description: It computes $\sum_{i=1}^{i=n} i^p$ It assumes C[n][k], inv[n] and pow(a, x)(a^x % mod) are defined.
Time: $\mathcal{O}(p)$ c8a978, 21 lines

```
void pre(){
  B[0] = 1;
  B[1] = -(mod+1)/2; //Bernoulli Nos
  for(int i = 2;i<51;i+=2){
    rep(k,i){
      B[i]+=(C[i+1][k]*B[k])%mod;
    }
    B[i]%=mod;
    B[i] = ((-B[i]*inv[i+1])%mod+mod)%mod;
  }
}
ll Faul(ll n,int p){
  ll ans = 0;
  repA(i,1,p+1){
    ll cur = B[p+1-i]*modpow(n,i)%mod;
    cur = cur*C[p+1][i]%mod;
    if(i==p) ans-=cur;
    else ans+=cur;
  }
  return ((ans%mod+mod)%mod)*modpow(p+1,mod
       -2)%mod;
}
```

5.2 Primality

eratosthenes.h

Description: Prime sieve for generating all primes up to a certain limit. isprime[i] is true iff i is a prime.
Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored. 29cd0a, 11 lines

```
const int MAX_PR = 5'000'000;
bitset<MAX_PR> isprime;
vi eratosthenes_sieve(int lim) {
  isprime.set(); isprime[0] = isprime[1] =
       0;
  for (int i = 4; i < lim; i += 2) isprime[i
       ] = 0;
  for (int i = 3; i*i < lim; i += 2) if (
       isprime[i])
    for (int j = i*i; j < lim; j += i*2)
       isprime[j] = 0;
  vi pr;
  rep(i,2,lim) if (isprime[i]) pr.push_back(
       i);
  return pr;
}
```

MillerRabin.h

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to 2^{64}; for larger numbers, extend A randomly.
Time: 7 times the complexity of $a^b \bmod c$.
"ModMulLL.h" 6ab8e1, 12 lines

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

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).
Time: $\mathcal{O}\left(n^{1/4}\right)$ gcd calls, less for numbers with small factors.
"ModMulLL.h", "MillerRabin.h" f5adaa, 18 lines

```
ull pollard(ull n) {
  auto f = [n](ull x) { return (mod_mul(x, x
       , n) + 1) % n; };
  if (!(n & 1)) return 2;
  for (ull i = 2;; i++) {
    ull x = i, y = f(x), p;
    while ((p = __gcd(n + y - x, n)) == 1)
      x = f(x), y = f(f(y));
    if (p != n) return p;
  }
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

5.3 Divisibility

euclid.h

Description: Finds two integers x and y, such that $ax + by = \gcd(a, b)$. If you just need gcd, use the built in `__gcd` instead. If a and b are coprime, then x is the inverse of $a \pmod b$. ee6239, 5 lines

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

CRT.h

Description: Chinese Remainder Theorem.
`crt(a, m, b, n)` computes x such that $x \equiv a \pmod m$, $x \equiv b \pmod n$. If $|a| < m$ and $|b| < n$, x will obey $0 \le x < \mathrm{lcm}(m, n)$. Assumes $mn < 2^{62}$.
Time: $\log(n)$
"euclid.h" 04d93a, 7 lines

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

5.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

phiFunction.h
Description: *Euler's* ϕ function is defined as $\phi(n) :=$ # of positive integers $\leq n$ that are coprime with n. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} ... (p_r - 1)p_r^{k_r - 1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.
Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p}\ \forall a$. cf7d6d, 8 lines

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

5.4 Fractions

ContinuedFractions.h
Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with $p, q \leq N$. It will obey $|p/q - x| \leq 1/qN$.
For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. ($p_k/q_k$ alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a's eventually become cyclic.
Time: $\mathcal{O}(\log N)$ dd6c5e, 21 lines

```
typedef double d; // for N ~ 1e7; long
    double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
  ll LP = 0, LQ = 1, P = 1, Q = 0, inf =
      LLONG_MAX; d y = x;
  for (;;) {
    ll lim = min(P ? (N-LP) / P : inf, Q ? (
        N-LQ) / Q : inf),
        a = (ll)floor(y), b = min(a, lim),
        NP = b*P + LP, NQ = b*Q + LQ;
    if (a > b) {
      // If b > a/2, we have a semi-
          convergent that gives us a
      // better approximation; if b = a/2,
          we *may* have one.
      // Return {P, Q} here for a more
          canonical approximation.
      return (abs(x - (d)NP / (d)NQ) < abs(x
           - (d)P / (d)Q)) ?
        make_pair(NP, NQ) : make_pair(P, Q);
    }
    if (abs(y = 1/(y - (d)a)) > 3*N) {
      return {NP, NQ};
    }
    LP = P; P = NP;
    LQ = Q; Q = NQ;
  }
}
```

FracBinarySearch.h
Description: Given f and N, finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \leq N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.
Usage: `fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}`
Time: $\mathcal{O}(\log(N))$ 27ab3e, 25 lines

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

5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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 $n < 1e19$.

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

IntPerm.h
Description: Permutation -> integer conversion. (Not order preserving.)
Time: $\mathcal{O}(n)$ e1b8ea, 6 lines

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

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 Burnside's lemma

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

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

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

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

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

6.2.2 Binomials

binomialModPrime.h

Description: Lucas' thm: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.

Time: $\mathcal{O}(\log_p n)$ 81845f, 10 lines

```
ll chooseModP(ll n, ll m, int p, vi& fact,
    vi& invfact) {
  ll c = 1;
  while (n || m) {
    ll a = n % p, b = m % p;
    if (a < b) return 0;
    c = c * fact[a] % p * invfact[b] % p *
        invfact[a - b] % p;
    n /= p; m /= p;
  }
  return c;
}
```

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.

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

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

6.3.3 Eulerian numbers

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

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 Bell numbers

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

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

6.3.6 Labeled unrooted trees

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

6.3.7 Catalan numbers

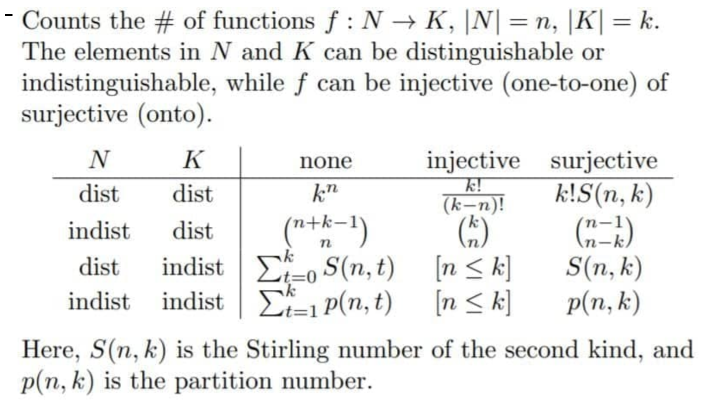
$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

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

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

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

6.3.8 Twelvefold way



- Counts the # of functions $f : N \to K$, $|N| = n$, $|K| = k$. The elements in N and K can be distinguishable or indistinguishable, while f can be injective (one-to-one) of surjective (onto).

N	K	none	injective	surjective
dist	dist	k^n	$\frac{k!}{(k-n)!}$	$k!S(n,k)$
indist	dist	$\binom{n+k-1}{n}$	$\binom{k}{n}$	$\binom{n-1}{n-k}$
dist	indist	$\sum_{t=0}^{k} S(n,t)$	$[n \leq k]$	$S(n,k)$
indist	indist	$\sum_{t=1}^{k} p(n,t)$	$[n \leq k]$	$p(n,k)$

Here, $S(n,k)$ is the Stirling number of the second kind, and $p(n,k)$ is the partition number.

Graph (7)

7.1 Fundamentals

BellmanFord.h

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

Time: $\mathcal{O}(VE)$ 53f20b, 23 lines

```
const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b
    ? a : -a; }};
struct Node { ll dist = inf; int prev = -1;
    };
void bellmanFord(vector<Node>& nodes, vector
    <Ed>& eds, int s) {
  nodes[s].dist = 0;
  sort(all(eds), [](Ed a, Ed b) { return a.s
      () < b.s(); });
  int lim = sz(nodes) / 2 + 2; // /3+100
      with shuffled vertices
  rep(i,0,lim) trav(ed, eds) {
    Node cur = nodes[ed.a], &dest = nodes[ed
        .b];
    if (abs(cur.dist) == inf) continue;
    ll d = cur.dist + ed.w;
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);
    }
  }
  rep(i,0,lim) trav(e, eds) {
    if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf;
  }
}
```

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j]$ = `inf` if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, `inf` if no path, or `-inf` if the path goes through a negative-weight cycle.

Time: $\mathcal{O}\left(N^3\right)$ 531245, 12 lines

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

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.

Time: $\mathcal{O}(|V| + |E|)$ 22a53d, 14 lines

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

7.2 Euler walk

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. 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, also put it->second in s (and then ret).

Time: $\mathcal{O}(E)$ where E is the number of edges. f8bd47, 27 lines

```
struct V {
  vector<pii> outs; // (dest, edge index)
  int nins = 0;
};
vi euler_walk(vector<V>& nodes, int nedges,
    int src=0) {
  int c = 0;
  trav(n, nodes) c += abs(n.nins - sz(n.outs
      ));
  if (c > 2) return {};
  vector<vector<pii>::iterator> its;
  trav(n, nodes)
    its.push_back(n.outs.begin());
  vector<bool> eu(nedges);
  vi ret, s = {src};
  while(!s.empty()) {
    int x = s.back();
    auto& it = its[x], end = nodes[x].outs.
        end();
    while(it != end && eu[it->second]) ++it;
    if(it == end) { ret.push_back(x); s.
        pop_back(); }
    else { s.push_back(it->first); eu[it->
        second] = true; }
  }
  if(sz(ret) != nedges+1)
    ret.clear(); // No Eulerian cycles/paths
  // else, non-cycle if ret.front() != ret.
      back()
  reverse(all(ret));
  return ret;
}
```

7.3 Network flow

PushRelabel.h

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

Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$ 6c4045, 49 lines

```
typedef ll Flow;
struct Edge {
  int dest, back;
  Flow f, c;
};
struct PushRelabel {
  vector<vector<Edge>> g;
  vector<Flow> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n),
      hs(2*n), H(n) {}
  void add_edge(int s, int t, Flow cap, Flow
       rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap})
        ;
  }
  void add_flow(Edge& e, Flow f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].
        push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest]
        -= f;
  }
  Flow maxflow(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    trav(e, g[s]) add_flow(e, e.c);
    for (int hi = 0;;) {
      while (hs[hi].empty()) if (!hi--)
          return -ec[s];
      int u = hs[hi].back(); hs[hi].pop_back
          ();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])
            ) {
          H[u] = 1e9;
          trav(e, g[u]) if (e.c && H[u] > H[
              e.dest]+1)
            H[u] = H[e.dest]+1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi <
              v)
            rep(i,0,v) if (hi < H[i] && H[i]
                < v)
              --co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[
            cur[u]->dest]+1)
          add_flow(*cur[u], min(ec[u], cur[u
              ]->c));
        else ++cur[u];
    }
  }
};
```

MinCostMaxFlow.h

Description: Min-cost max-flow. cap[i][j] != cap[j][i] is allowed; double edges are not. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.

Time: Approximately $\mathcal{O}\left(E^2\right)$ 6915ce, 81 lines

```
#include <bits/extc++.h>
const ll INF = numeric_limits<ll>::max() /
    4;
typedef vector<ll> VL;
struct MCMF {
  int N;
  vector<vi> ed, red;
  vector<VL> cap, flow, cost;
  vi seen;
  VL dist, pi;
  vector<pii> par;
  MCMF(int N) :
    N(N), ed(N), red(N), cap(N, VL(N)), flow
        (cap), cost(cap),
    seen(N), dist(N), pi(N), par(N) {}
  void addEdge(int from, int to, ll cap, ll
      cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
    ed[from].push_back(to);
    red[to].push_back(from);
  }
  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;
    __gnu_pbds::priority_queue<pair<ll, int
        >> q;
    vector<decltype(q)::point_iterator> its(
        N);
    q.push({0, s});
    auto relax = [&](int i, ll cap, ll cost,
        int dir) {
      ll val = di - pi[i] + cost;
      if (cap && val < dist[i]) {
        dist[i] = val;
        par[i] = {s, dir};
        if (its[i] == q.end()) its[i] = q.
            push({-dist[i], i});
        else q.modify(its[i], {-dist[i], i})
            ;
      }
    };
    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      trav(i, ed[s]) if (!seen[i])
        relax(i, cap[s][i] - flow[s][i],
            cost[s][i], 1);
      trav(i, red[s]) if (!seen[i])
        relax(i, flow[i][s], -cost[i][s], 0)
            ;
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i],
        INF);
  }
  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (int p,r,x = t; tie(p,r) = par[x],
          x != s; x = p)
        fl = min(fl, r ? cap[p][x] - flow[p
            ][x] : flow[x][p]);
      totflow += fl;
      for (int p,r,x = t; tie(p,r) = par[x],
          x != s; x = p)
        if (r) flow[p][x] += fl;
        else flow[x][p] -= fl;
    }
    rep(i,0,N) rep(j,0,N) totcost += cost[i
        ][j] * flow[i][j];
    return {totflow, totcost};
  }
  // If some costs can be negative, call
      this before maxflow:
  void setpi(int s) { // (otherwise, leave
      this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INF)
        trav(to, ed[i]) if (cap[i][to])
          if ((v = pi[i] + cost[i][to]) < pi
              [to])
            pi[to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
  }
};
```

EdmondsKarp.h

Description: Flow algorithm with guaranteed complexity $O(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only.

979bb9, 35 lines

```
template<class T> T edmondsKarp(vector<
    unordered_map<int, T>>& graph, int
    source, int sink) {
  assert(source != sink);
  T flow = 0;
  vi par(sz(graph)), q = par;
  for (;;) {
    fill(all(par), -1);
    par[source] = 0;
    int ptr = 1;
    q[0] = source;
    rep(i,0,ptr) {
      int x = q[i];
      trav(e, graph[x]) {
        if (par[e.first] == -1 && e.second >
            0) {
          par[e.first] = x;
          q[ptr++] = e.first;
          if (e.first == sink) goto out;
        }
      }
    }
    return flow;
out:
    T inc = numeric_limits<T>::max();
    for (int y = sink; y != source; y = par[
        y])
      inc = min(inc, graph[par[y]][y]);
    flow += inc;
    for (int y = sink; y != source; y = par[
        y]) {
      int p = par[y];
      if ((graph[p][y] -= inc) <= 0) graph[p
          ].erase(y);
      graph[y][p] += inc;
    }
  }
}
```

MinCut.h

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

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}\left(V^3\right)$

03261f, 31 lines

```
pair<int, vi> GetMinCut(vector<vi>& weights)
    {
  int N = sz(weights);
  vi used(N), cut, best_cut;
  int best_weight = -1;
  for (int phase = N-1; phase >= 0; phase--)
      {
    vi w = weights[0], added = used;
    int prev, k = 0;
    rep(i,0,phase){
      prev = k;
      k = -1;
      rep(j,1,N)
        if (!added[j] && (k == -1 || w[j] >
            w[k])) k = j;
      if (i == phase-1) {
        rep(j,0,N) weights[prev][j] +=
            weights[k][j];
        rep(j,0,N) weights[j][prev] =
            weights[prev][j];
        used[k] = true;
        cut.push_back(k);
        if (best_weight == -1 || w[k] <
            best_weight) {
          best_cut = cut;
          best_weight = w[k];
        }
      } else {
        rep(j,0,N)
          w[j] += weights[k][j];
        added[k] = true;
      }
    }
  }
  return {best_weight, best_cut};
}
```

7.4 Matching

hopcroftKarp.h

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

536939, 42 lines

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

DFSMatching.h

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

6a3472, 22 lines

```
bool find(int j, vector<vi>& g, vi& btoa, vi
    & vis) {
```

```
  if (btoa[j] == -1) return 1;
  vis[j] = 1; int di = btoa[j];
  trav(e, g[di])
    if (!vis[e] && find(e, g, btoa, vis)) {
      btoa[e] = di;
      return 1;
    }
  return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
  vi vis;
  rep(i,0,sz(g)) {
    vis.assign(sz(btoa), 0);
    trav(j,g[i])
      if (find(j, g, btoa, vis)) {
        btoa[j] = i;
        break;
      }
  }
  return sz(btoa) - (int)count(all(btoa),
     -1);
}
```

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" d0b3f2, 20 lines

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

WeightedMatching.h

Description: Min cost bipartite matching. Negate costs for max cost.
Time: $\mathcal{O}(N^3)$ 055ca9, 75 lines

```
typedef vector<double> vd;
bool zero(double x) { return fabs(x) < 1e
   -10; }
double MinCostMatching(const vector<vd>&
   cost, vi& L, vi& R) {
  int n = sz(cost), mated = 0;
  vd dist(n), u(n), v(n);
  vi dad(n), seen(n);
  rep(i,0,n) {
    u[i] = cost[i][0];
    rep(j,1,n) u[i] = min(u[i], cost[i][j]);
  }
  rep(j,0,n) {
    v[j] = cost[0][j] - u[0];
    rep(i,1,n) v[j] = min(v[j], cost[i][j] -
       u[i]);
  }
  L = R = vi(n, -1);
  rep(i,0,n) rep(j,0,n) {
    if (R[j] != -1) continue;
    if (zero(cost[i][j] - u[i] - v[j])) {
      L[i] = j;
      R[j] = i;
      mated++;
      break;
    }
  }
  for (; mated < n; mated++) { // until
     solution is feasible
    int s = 0;
    while (L[s] != -1) s++;
    fill(all(dad), -1);
    fill(all(seen), 0);
    rep(k,0,n)
      dist[k] = cost[s][k] - u[s] - v[k];
    int j = 0;
    for (;;) {
      j = -1;
      rep(k,0,n){
        if (seen[k]) continue;
        if (j == -1 || dist[k] < dist[j]) j
           = k;
      }
      seen[j] = 1;
      int i = R[j];
      if (i == -1) break;
      rep(k,0,n) {
        if (seen[k]) continue;
        auto new_dist = dist[j] + cost[i][k]
           - u[i] - v[k];
        if (dist[k] > new_dist) {
          dist[k] = new_dist;
          dad[k] = j;
        }
      }
    }
    rep(k,0,n) {
      if (k == j || !seen[k]) continue;
      auto w = dist[k] - dist[j];
      v[k] += w, u[R[k]] -= w;
    }
    u[s] += dist[j];
    while (dad[j] >= 0) {
      int d = dad[j];
      R[j] = R[d];
      L[R[j]] = j;
      j = d;
    }
    R[j] = s;
    L[s] = j;
  }
  auto value = vd(1)[0];
  rep(i,0,n) value += cost[i][L[i]];
  return value;
}
```

GeneralMatching.h

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

"../numerical/MatrixInverse-mod.h" bb8be4, 40 lines

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

7.5 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: $\mathcal{O}(E + V)$ bb2963, 24 lines

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

BiconnectedComponents.h

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

Usage: `int eid = 0; ed.resize(N);`
```
for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) {...});
```
Time: $\mathcal{O}(E+V)$ cca7e6, 33 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
  int me = num[at] = ++Time, e, y, top = me;
  trav(pa, ed[at]) if (pa.second != par) {
    tie(y, e) = pa;
    if (num[y]) {
      top = min(top, num[y]);
      if (num[y] < me)
        st.push_back(e);
    } else {
      int si = sz(st);
      int up = dfs(y, e, f);
      top = min(top, up);
      if (up == me) {
        st.push_back(e);
        f(vi(st.begin() + si, st.end()));
        st.resize(si);
      }
      else if (up < me) st.push_back(e);
      else { /* e is a bridge */ }
    }
  }
  return top;
}
template<class F>
void bicomps(F f) {
  num.assign(sz(ed), 0);
  rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f)
     ;
}
```

2sat.h

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

Usage: `TwoSat ts(number of boolean variables);`
```
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.set_value(2); // Var 2 is true
ts.at_most_one({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. 0911c1, 56 lines

```
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false, 1 = true
  TwoSat(int n = 0) : N(n), gr(2*n) {}
  int add_var() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }
  void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1);
    gr[j].push_back(f^1);
  }
  void set_value(int x) { either(x, x); }
  void at_most_one(const vi& li) { // (
     optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = add_var();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }
  vi val, comp, z; int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x; z.
       push_back(i);
    trav(e, gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = low;
      if (values[x>>1] == -1)
        values[x>>1] = x&1;
    } while (x != i);
    return val[i] = low;
  }
  bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1])
       return 0;
    return 1;
  }
};
```

7.6 Heuristics

MaximalCliques.h

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

Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs b0d5b1, 12 lines

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

MaximumClique.h

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.

Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs. fbbef1, 49 lines

```
typedef vector<bitset<200>> vb;
struct Maxclique {
  double limit=0.025, pk=0;
  struct Vertex { int i, d=0; };
  typedef vector<Vertex> vv;
  vb e;
  vv V;
  vector<vi> C;
  vi qmax, q, S, old;
  void init(vv& r) {
    trav(v,r) v.d = 0;
    trav(v, r) trav(j, r) v.d += e[v.i][j.i
       ];
    sort(all(r), [](auto a, auto b) { return
        a.d > b.d; });
    int mxD = r[0].d;
    rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
      if (sz(q) + R.back().d <= sz(qmax))
        return;
      q.push_back(R.back().i);
      vv T;
      trav(v,R) if (e[R.back().i][v.i]) T.
         push_back({v.i});
      if (sz(T)) {
        if (S[lev]++ / ++pk < limit) init(T)
           ;
        int j = 0, mxk = 1, mnk = max(sz(
           qmax) - sz(q) + 1, 1);
        C[1].clear(), C[2].clear();
        trav(v, T) {
          int k = 1;
          auto f = [&](int i) { return e[v.i
             ][i]; };
          while (any_of(all(C[k]), f)) k++;
          if (k > mxk) mxk = k, C[mxk + 1].
             clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        if (j > 0) T[j - 1].d = 0;
        rep(k,mnk,mxk + 1) trav(i, C[k])
          T[j].i = i, T[j++].d = k;
        expand(T, lev + 1);
      } else if (sz(q) > sz(qmax)) qmax = q;
      q.pop_back(), R.pop_back();
    }
  }
  vi maxClique() { init(V), expand(V);
     return qmax; }
  Maxclique(vb conn) : e(conn), C(sz(e)+1),
      S(sz(C)), old(S) {
    rep(i,0,sz(e)) V.push_back({i});
  }
};
```

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.~~

7.7 Trees

TreePower.h

Description: Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.

Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$ bfce85, 25 lines

```
vector<vi> treeJump(vi& P){
  int on = 1, d = 1;
  while(on < sz(P)) on *= 2, d++;
  vector<vi> jmp(d, P);
  rep(i,1,d) rep(j,0,sz(P))
    jmp[i][j] = jmp[i-1][jmp[i-1][j]];
  return jmp;
}
int jmp(vector<vi>& tbl, int nod, int steps)
    {
  rep(i,0,sz(tbl))
    if(steps&(1<<i)) nod = tbl[i][nod];
  return nod;
}
int lca(vector<vi>& tbl, vi& depth, int a,
    int b) {
  if (depth[a] < depth[b]) swap(a, b);
  a = jmp(tbl, a, depth[a] - depth[b]);
  if (a == b) return a;
  for (int i = sz(tbl); i--;) {
    int c = tbl[i][a], d = tbl[i][b];
    if (c != d) a = c, b = d;
  }
  return tbl[0][a];
}
```

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. Can also find the distance between two nodes.

Usage: `LCA lca(undirGraph);`
`lca.query(firstNode, secondNode);`
`lca.distance(firstNode, secondNode);`

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

"../data-structures/RMQ.h" aa0d4d, 37 lines

```
typedef vector<pii> vpi;
typedef vector<vpi> graph;
struct LCA {
  vi time;
  vector<ll> dist;
  RMQ<pii> rmq;
  LCA(graph& C) : time(sz(C), -99), dist(sz(
      C)), rmq(dfs(C)) {}
  vpi dfs(graph& C) {
    vector<tuple<int, int, int, ll>> q(1);
    vpi ret;
    int T = 0, v, p, d; ll di;
    while (!q.empty()) {
      tie(v, p, d, di) = q.back();
      q.pop_back();
      if (d) ret.emplace_back(d, p);
      time[v] = T++;
      dist[v] = di;
      trav(e, C[v]) if (e.first != p)
        q.emplace_back(e.first, v, d+1, di +
            e.second);
    }
    return ret;
  }
  int query(int a, int b) {
    if (a == b) return a;
    a = time[a], b = time[b];
    return rmq.query(min(a, b), max(a, b)).
        second;
  }
  ll distance(int a, int b) {
    int lca = query(a, b);
    return dist[a] + dist[b] - 2 * dist[lca
        ];
  }
};
```

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" dabd75, 20 lines

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

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges. The function of the HLD can be changed by modifying T, LOW and f. f is assumed to be associative and commutative.

Usage: `HLD hld(G);`
`hld.update(index, value);`
`tie(value, lca) = hld.query(n1, n2);`

"../data-structures/SegmentTree.h" d952a9, 93 lines

```
typedef vector<pii> vpi;
struct Node {
  int d, par, val, chain = -1, pos = -1;
};
struct Chain {
  int par, val;
  vector<int> nodes;
  Tree tree;
};
struct HLD {
  typedef int T;
  const T LOW = -(1<<29);
  void f(T& a, T b) { a = max(a, b); }
  vector<Node> V;
  vector<Chain> C;
  HLD(vector<vpi>& g) : V(sz(g)) {
    dfs(0, -1, g, 0);
    trav(c, C) {
      c.tree = {sz(c.nodes), 0};
      for (int ni : c.nodes)
        c.tree.update(V[ni].pos, V[ni].val);
    }
  }
  void update(int node, T val) {
    Node& n = V[node]; n.val = val;
    if (n.chain != -1) C[n.chain].tree.
        update(n.pos, val);
  }
  int pard(Node& nod) {
    if (nod.par == -1) return -1;
    return V[nod.chain == -1 ? nod.par : C[
        nod.chain].par].d;
  }
  // query all *edges* between n1, n2
  pair<T, int> query(int i1, int i2) {
    T ans = LOW;
    while(i1 != i2) {
      Node n1 = V[i1], n2 = V[i2];
      if (n1.chain != -1 && n1.chain == n2.
          chain) {
        int lo = n1.pos, hi = n2.pos;
        if (lo > hi) swap(lo, hi);
        f(ans, C[n1.chain].tree.query(lo, hi
            ));
        i1 = i2 = C[n1.chain].nodes[hi];
      } else {
        if (pard(n1) < pard(n2))
          n1 = n2, swap(i1, i2);
        if (n1.chain == -1)
          f(ans, n1.val), i1 = n1.par;
        else {
          Chain& c = C[n1.chain];
          f(ans, n1.pos ? c.tree.query(n1.
              pos, sz(c.nodes))
                        : c.tree.s[1]);
          i1 = c.par;
        }
      }
    }
    return make_pair(ans, i1);
  }
  // query all *nodes* between n1, n2
  pair<T, int> query2(int i1, int i2) {
    pair<T, int> ans = query(i1, i2);
    f(ans.first, V[ans.second].val);
    return ans;
  }
  pii dfs(int at, int par, vector<vpi>& g,
      int d) {
    V[at].d = d; V[at].par = par;
    int sum = 1, ch, nod, sz;
    tuple<int,int,int> mx(-1,-1,-1);
    trav(e, g[at]){
      if (e.first == par) continue;
      tie(sz, ch) = dfs(e.first, at, g, d+1)
          ;
      V[e.first].val = e.second;
      sum += sz;
      mx = max(mx, make_tuple(sz, e.first,
          ch));
    }
    tie(sz, nod, ch) = mx;
    if (2*sz < sum) return pii(sum, -1);
    if (ch == -1) { ch = sz(C); C.
      emplace_back(); }
    V[nod].pos = sz(C[ch].nodes);
    V[nod].chain = ch;
    C[ch].par = at;
    C[ch].nodes.push_back(nod);
    return pii(sum, ch);
  }
};
```

CentroidDecomposition.h

Description: It finds the centroid tree for given tree. par[x] denotes centroid parent of x. Initialize global variables correctly before using. 147161, 34 lines

```
vector<int> adj[maxN+1];
vi sz, par;
vector<bool> block;
int nn;
void dfs0(int x, int p) {
  nn++;
  sz[x] = 1;
  for(auto v: adj[x]) {
    if(v!=p && !block[v]) {
      dfs0(v,x);
      sz[x] += sz[v];
    }
  }
}
int dfs1(int x, int p) {
  for(auto v: adj[x]) {
    if(v!=p && !block[v] && sz[v]>nn/2)
      return dfs1(v, x);
  }
  return x;
}
// Parent of root is parent only
void decompose(int root, int p) {
  nn=0;
  dfs0(root, root);
  int centroid = dfs1(root, root);
  if(p==-1) p = centroid;
  par[centroid] = p;
  block[centroid] = true;
  for(auto v: adj[centroid]) {
    if(!block[v])
      decompose(v, centroid);
  }
}
```

LinkCutTree.h

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.
Time: All operations take amortized $\mathcal{O}(\log N)$. 693483, 90 lines

```
struct Node { // Splay tree. Root's pp
   contains tree's parent.
  Node *p = 0, *pp = 0, *c[2];
  bool flip = 0;
  Node() { c[0] = c[1] = 0; fix(); }
  void fix() {
    if (c[0]) c[0]->p = this;
    if (c[1]) c[1]->p = this;
    // (+ update sum of subtree elements etc
        . if wanted)
  }
  void push_flip() {
    if (!flip) return;
    flip = 0; swap(c[0], c[1]);
    if (c[0]) c[0]->flip ^= 1;
    if (c[1]) c[1]->flip ^= 1;
  }
  int up() { return p ? p->c[1] == this :
     -1; }
  void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h
       ], *z = b ? y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
      x->c[h] = y->c[h ^ 1];
      z->c[h ^ 1] = b ? x : this;
    }
    y->c[i ^ 1] = b ? this : x;
    fix(); x->fix(); y->fix();
    if (p) p->fix();
    swap(pp, y->pp);
  }
  void splay() {
    for (push_flip(); p; ) {
      if (p->p) p->p->push_flip();
      p->push_flip(); push_flip();
      int c1 = up(), c2 = p->up();
      if (c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
  Node* first() {
    push_flip();
    return c[0] ? c[0]->first() : (splay(),
        this);
  }
};
struct LinkCut {
  vector<Node> node;
  LinkCut(int N) : node(N) {}
  void link(int u, int v) { // add an edge (
     u, v)
    assert(!connected(u, v));
    make_root(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge
     (u, v)
    Node *x = &node[u], *top = &node[v];
    make_root(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) { // are u, v
      in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void make_root(Node* u) {
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
  Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp;
          }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
};
```

DirectedMST.h

Description: Edmonds' algorithm for finding the weight of the 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/UnionFind.h" a69883, 48 lines

```
struct Edge { int a, b; ll w; };
struct Node {
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a
   ->l, a->r); }
ll dmst(int n, int r, vector<Edge>& g) {
  UF uf(n);
  vector<Node*> heap(n);
  trav(e, g) heap[e.b] = merge(heap[e.b],
      new Node{e});
  ll res = 0;
  vi seen(n, -1), path(n);
  seen[r] = r;
  rep(s,0,n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      path[qi++] = u, seen[u] = s;
      if (!heap[u]) return -1;
      Edge e = heap[u]->top();
      heap[u]->delta -= e.w, pop(heap[u]);
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) {
        Node* cyc = 0;
        do cyc = merge(cyc, heap[w = path[--
            qi]]);
        while (uf.join(u, w));
        u = uf.find(u);
        heap[u] = cyc, seen[u] = -1;
      }
    }
  }
  return res;
}
```

MatrixTree.h

Description: To count the number of spanning trees in an undirected graph G: create an $N \times N$ matrix `mat`, and for each edge $(a, b) \in G$, do `mat[a][a]++, mat[b][b]++, mat[a][b]--, mat[b][a]--`. Remove the ~~last row and column, and take the determinant.~~

Geometry (8)

8.1 Geometric primitives

Point.h

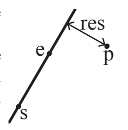
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

```
template <class T> int sgn(T x) { return (x
    > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y)
      {}
  bool operator<(P p) const { return tie(x,y
      ) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,
      y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y
      +p.y); }
  P operator-(P p) const { return P(x-p.x, y
      -p.y); }
  P operator*(T d) const { return P(x*d, y*d
      ); }
  P operator/(T d) const { return P(x/d, y/d
      ); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x;
      }
  T cross(P a, P b) const { return (a-*this)
      .cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)
      dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x);
      }
  P unit() const { return *this/dist(); } //
       makes dist()=1
  P perp() const { return P(-y, x); } //
      rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw
      around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*
        cos(a)); }
  friend ostream& operator<<(ostream& os, P
      p) {
    return os << "(" << p.x << "," << p.y <<
         ")"; }
};
```

lineDistance.h

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

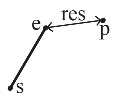


"Point.h" f6bf6b, 4 lines

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

SegmentDistance.h

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



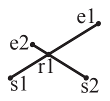
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" 5c88f4, 6 lines

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

SegmentIntersection.h

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



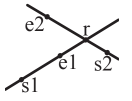
Usage: `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

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

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

lineIntersection.h

Description:
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.



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

"Point.h" a01f81, 8 lines

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

sideOf.h

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

"Point.h" 3af81c, 9 lines

```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.
    cross(e, p)); }
template<class P>
int sideOf(const P& s, const P& e, const P&
    p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

OnSegment.h

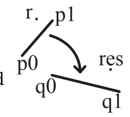
Description: Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

"Point.h" c597e8, 3 lines

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

linearTransformation.h

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



"Point.h" 03a306, 6 lines

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

Angle.h

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

Usage: `vector<Angle> v = {w[0], w[0].t360() ...}; // sorted`
`int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }`
`// sweeps j such that (j-i) represents the number of positively`
`oriented triangles with vertices at 0 and i` 0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y),
      t(t) {}
  Angle operator-(Angle b) const { return {x
      -b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (
      half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t +
      half()}; }
  Angle t360() const { return {x, y, t + 1};
      }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also
      compare distances
  return make_tuple(a.t, a.half(), a.y * (ll
      )b.x) <
         make_tuple(b.t, b.half(), a.x * (ll
             )b.y);
}
// Given two points, this calculates the
    smallest angle between
// them, i.e., the angle that covers the
    defined line segment.
pair<Angle, Angle> segmentAngles(Angle a,
    Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.
              t360()));
}
Angle operator+(Angle a, Angle b) { // point
    a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle
    b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b
      .x, tu - (b < a)};
}
```

8.2 Circles

CircleIntersection.h

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

`"Point.h"` 84d6d3, 11 lines

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

CircleTangents.h

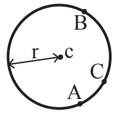
Description: Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second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

`"Point.h"` b0153d, 13 lines

```
template<class P>
vector<pair<P, P>> tangents(P c1, double r1,
    P c2, double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 =
      d2 - dr * dr;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) *
        sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2
        });
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

circumcircle.h

Description:



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

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

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

MinimumEnclosingCircle.h

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

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

`"circumcircle.h"` 09dd0a, 17 lines

```
pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r
      * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r *
        EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r
          * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

8.3 Polygons

InsidePolygon.h

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

Usage: `vector<P> v = {P{4,4}, P{1,2}, P{2,1}};`
`bool in = inPolygon(v, P{3, 3}, false);`

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

`"Point.h", "OnSegment.h", "SegmentDistance.h"` 2bf504, 11 lines

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

PolygonArea.h

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

`"Point.h"` f12300, 6 lines

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

PolygonCenter.h

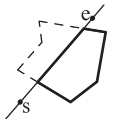
Description: Returns the center of mass for a polygon.

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

`"Point.h"` 9706dc, 9 lines

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

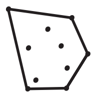
PolygonCut.h
Description:
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`



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

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

ConvexHull.h
Description:
Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.



Time: $\mathcal{O}(n \log n)$
"Point.h" 26a0a9, 13 lines

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(
      all(pts)))
    trav(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/colinear points).
c571b8, 12 lines

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

PointInsideHull.h
Description: Determine whether a point t lies inside a convex hull (CCW order, with no colinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$
"Point.h", "sideOf.h", "OnSegment.h" 71446b, 14 lines

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

LineHullIntersection.h
Description: Line-convex polygon intersection. The polygon must be ccw and have no colinear 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: $\mathcal{O}(N + Q \log n)$
"Point.h" 758f22, 39 lines

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

8.4 Misc. Point Set Problems

ClosestPair.h
Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$
"Point.h" d31bbf, 17 lines

```
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;
  trav(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;
}
```

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 the box is wider than
          high (not best heuristic...)
      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);
  }
};
```

FastDelaunay.h

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

"Point.h" bf87ec, 88 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 {
  bool mark; Q o, rot; P p;
  P F() { return r()->p; }
  Q r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
};
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 q[] = {new Quad{0,0,0,orig}, new Quad
      {0,0,0,arb},
           new Quad{0,0,0,dest}, new Quad
               {0,0,0,arb}};
  rep(i,0,4)
    q[i]->o = q[-i & 3], q[i]->rot = q[(i+1)
        & 3];
  return *q;
}
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 = 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.

1ec4d3, 6 lines

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

3dHull.h

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

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

"Point3D.h" c172e9, 49 lines

```
typedef Point3D<double> P3;
struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1); }
  int a, b;
};
struct F { P3 q; int a, b, c; };
vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);
  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  trav(it, FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
  return FS;
};
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis. 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

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

d4375c, 16 lines

```
vi pi(const string& s) {
  vi p(sz(s));
  rep(i,1,sz(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}
vi match(const string& s, const string& pat) {
  vi p = pi(pat + '\0' + s), res;
  rep(i,sz(p)-sz(s),sz(p))
    if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
  return res;
}
```

Zfunc.h

Description: z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)

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

3ae526, 12 lines

```
vi Z(string S) {
  vi z(sz(S));
  int l = -1, r = -1;
  rep(i,1,sz(S)) {
    z[i] = i >= r ? 0 : min(r - i, z[i - l]);
    while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
      z[i]++;
    if (i + z[i] > r)
      l = i, r = i + z[i];
  }
  return z;
}
```

Manacher.h

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

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

e7ad79, 13 lines

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

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

Usage: `rotate(v.begin(), v.begin()+min_rotation(v), v.end());`

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

358164, 8 lines

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

SuffixArray.h

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

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

38db9f, 23 lines

```
struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string& s, int lim=256) { // or basic_string<int>
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
    sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
      p = j, iota(all(y), n - j);
      rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i,0,n) ws[x[i]]++;
      rep(i,1,lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i
          ]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i,1,n) a = sa[i - 1], b = sa[i], x
          [b] =
        (y[a] == y[b] && y[a + j] == y[b + j
            ]) ? p - 1 : p++;
    }
    rep(i,1,n) rank[sa[i]] = i;
    for (int i = 0, j; i < n - 1; lcp[rank[i
        ++]] = k)
      for (k && k--, j = sa[rank[i] - 1];
          s[i + k] == s[j + k]; k++);
  }
};
```

SuffixTree.h

Description: Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r) into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r) substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).

Time: $\mathcal{O}(26N)$ aae0b8, 50 lines

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

PalindromicTree.h

Description: Finds all disctint palindromic substrings. T.display(s) prints all substrings.

Time: $\mathcal{O}(n)$ b8f648, 54 lines

```
struct palindromic_tree {
  vector<vector<int>> next;
  vector<int> suf, len;
  int new_node() {
    next.push_back(vector<int>(256,-1));
    suf.push_back(0);
    len.push_back(0);
    return next.size() - 1;
  }
  palindromic_tree(char *s) {
    len[new_node()] = -1;
    len[new_node()] = 0;
    int t = 1;
    for (int i = 0; s[i]; ++i) {
      int p = t;
      for (; i-1-len[p] < 0 || s[i-1-len
          [p]] != s[i]; p = suf[p]);
      if ((t = next[p][s[i]]) >= 0)
        continue;
      t = new_node();
      len[t] = len[p] + 2;
      next[p][s[i]] = t;
      if (len[t] == 1) {
        suf[t] = 1; // EMPTY
      } else {
        p = suf[p];
        for (; i-1-len[p] < 0 || s[i-1-
            len[p]] != s[i]; p = suf[p]);
        suf[t] = next[p][s[i]];
      }
    }
  }
  void display() {
    vector<char> buf;
    function<void (int)> rec = [&](int p
        ) {
      if (len[p] > 0) {
        for (int i = buf.size()-1; i >=
            0; --i) cout << buf[i];
        for (int i = len[p] % 2; i < buf
            .size(); ++i) cout << buf[i];
        cout << endl;
      }
      for (int a = 0; a < 256; ++a) {
        if (next[p][a] >= 0) {
          buf.push_back(a);
          rec(next[p][a]);
          buf.pop_back();
        }
      }
    };
    rec(0); rec(1);
  }
};
```

Hashing.h

Description: Self-explanatory methods for string hashing. acb5db, 44 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod
    2^64 and more
// code, but works on evil test data (e.g.
    Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash
    the same mod 2^64).
// "typedef ull H;" instead if you think
    test data is random,
// or work mod 10^9+7 if the Birthday
    paradox is not a problem.
struct H {
  typedef uint64_t ull;
  ull x; H(ull x=0) : x(x) {}
#define OP(O,A,B) H operator O(H o) { ull r
    = x; asm \
  (A "addq %%rdx, %0\n adcq $0,%0" : "+a"(r)
      : B); return r; }
  OP(+,,"d"(o.x)) OP(*,"mul %1\n", "r"(o.x)
      : "rdx")
  H operator-(H o) { return *this + ~o.x; }
  ull get() const { return x + !~x; }
  bool operator==(H o) const { return get()
      == o.get(); }
  bool operator<(H o) const { return get() <
      o.get(); }
};
static const H C = (ll)1e11+3; // (order ~ 3
    e9; random also ok)
struct HashInterval {
  vector<H> ha, pw;
  HashInterval(string& str) : ha(sz(str)+1),
      pw(ha) {
    pw[0] = 1;
    rep(i,0,sz(str))
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
  H hashInterval(int a, int b) { // hash [a,
      b)
    return ha[b] - ha[a] * pw[b - a];
  }
};
vector<H> getHashes(string& str, int length)
    {
  if (sz(str) < length) return {};
  H h = 0, pw = 1;
  rep(i,0,length)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h};
  rep(i,length,sz(str)) {
    ret.push_back(h = h * C + str[i] - pw *
        str[i-length]);
  }
  return ret;
}
H hashString(string& s) { H h{}; trav(c,s) h
    =h*C+c; return h; }
```

AhoCorasick.h

Description: Aho-Corasick tree is used for multiple pattern matching. Initialize the tree with create(patterns). find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(_, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input.
Time: create is $\mathcal{O}(26N)$ where N is the sum of length of patterns. find is $\mathcal{O}(M)$ where M is the length of the word. findAll is $\mathcal{O}(NM)$. 716ac4, 67 lines

```
struct AhoCorasick {
  enum {alpha = 26, first = 'A'};
  struct Node {
    // (nmatches is optional)
    int back, next[alpha], start = -1, end =
        -1, nmatches = 0;
    Node(int v) { memset(next, v, sizeof(
        next)); }
  };
  vector<Node> N;
  vector<int> backp;
  void insert(string& s, int j) {
    assert(!s.empty());
    int n = 0;
    trav(c, s) {
      int& m = N[n].next[c - first];
      if (m == -1) { n = m = sz(N); N.
          emplace_back(-1); }
      else n = m;
    }
    if (N[n].end == -1) N[n].start = j;
    backp.push_back(N[n].end);
    N[n].end = j;
    N[n].nmatches++;
  }
  AhoCorasick(vector<string>& pat) {
    N.emplace_back(-1);
    rep(i,0,sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);
    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
      int n = q.front(), prev = N[n].back;
      rep(i,0,alpha) {
        int &ed = N[n].next[i], y = N[prev].
            next[i];
        if (ed == -1) ed = y;
        else {
          N[ed].back = y;
          (N[ed].end == -1 ? N[ed].end :
              backp[N[ed].start])
            = N[y].end;
          N[ed].nmatches += N[y].nmatches;
          q.push(ed);
        }
      }
    }
  }
  vi find(string word) {
    int n = 0;
    vi res; // ll count = 0;
    trav(c, word) {
      n = N[n].next[c - first];
      res.push_back(N[n].end);
      // count += N[n].nmatches;
    }
    return res;
  }
  vector<vi> findAll(vector<string>& pat,
      string word) {
    vi r = find(word);
    vector<vi> res(sz(word));
    rep(i,0,sz(word)) {
      int ind = r[i];
      while (ind != -1) {
        res[i - sz(pat[ind]) + 1].push_back(
            ind);
        ind = backp[ind];
      }
    }
    return res;
  }
};
```

Various (10)

10.1 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$ edce47, 23 lines

```
set<pii>::iterator addInterval(set<pii>& is,
    int L, int R) {
  if (L == R) return is.end();
  auto it = is.lower_bound({L, R}), before =
      it;
  while (it != is.end() && it->first <= R) {
    R = max(R, it->second);
    before = it = is.erase(it);
  }
  if (it != is.begin() && (--it)->second >=
      L) {
    L = min(L, it->first);
    R = max(R, it->second);
    is.erase(it);
  }
  return is.insert(before, {L,R});
}
void removeInterval(set<pii>& is, int L, int
    R) {
  if (L == R) return;
  auto it = addInterval(is, L, R);
  auto r2 = it->second;
  if (it->first == L) is.erase(it);
  else (int&)it->second = L;
  if (R != r2) is.emplace(R, r2);
}
```

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

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I)
    {
  vi S(sz(I)), R;
  iota(all(S), 0);
  sort(all(S), [&](int a, int b) { return I[
      a] < I[b]; });
  T cur = G.first;
  int at = 0;
  while (cur < G.second) { // (A)
    pair<T, int> mx = make_pair(cur, -1);
    while (at < sz(I) && I[S[at]].first <=
        cur) {
      mx = max(mx, make_pair(I[S[at]].second
          , S[at]));
      at++;
    }
    if (mx.second == -1) return {};
    cur = mx.first;
    R.push_back(mx.second);
  }
  return R;
}
```

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$ 753a4c, 19 lines

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

10.2 Misc. algorithms

TernarySearch.h

Description: 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, 13 lines

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

LIS.h

Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$ 5b77eb, 17 lines

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

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

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

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  ll f(int ind, int k) { return dp[ind][k];
      }
  void store(int ind, int k, ll v) { res[ind
      ] = pii(k, v); }
  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
      best = min(best, make_pair(f(mid, k),
          k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R,
      INT_MIN, INT_MAX); }
};
```

10.4 Debugging tricks

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

10.5 Optimization tricks

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 for loops and optimizes floating points better (assumes associativity and turns off denormals).
- `#pragma GCC target ("avx,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 4 times faster than usual, where b is constant but not known at compile time. Fails for $b = 1$. c977c5, 10 lines

```
typedef unsigned long long ull;
typedef __uint128_t L;
struct FastMod {
  ull b, m;
  FastMod(ull b) : b(b), m(ull((L(1) << 64)
      / b)) {}
  ull reduce(ull a) {
    ull q = (ull)((L(m) * a) >> 64), r = a -
        q * b;
    return r >= b ? r - b : r;
  }
};
```

BumpAllocator.h

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

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

SmallPtr.h

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

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

BumpAllocatorSTL.h

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

```
char buf[450 << 20] alignas(16);
size_t buf_ind = sizeof buf;
template<class T> struct small {
  typedef T value_type;
  small() {}
  template<class U> small(const U&) {}
  T* allocate(size_t n) {
    buf_ind -= n * sizeof(T);
    buf_ind &= 0 - alignof(T);
    return (T*)(buf + buf_ind);
  }
  void deallocate(T*, size_t) {}
};
```

Unrolling.h

520e76, 5 lines

```
#define F {...; ++i;}
int i = from;
while (i&3 && i < to) F // for alignment, if
    needed
while (i + 4 <= to) { F F F F }
while (i < to) F
```