BRACU Crow's Library

newKACTL version 1 November, 2024

BRACU Crows

Contents

Contest (1)

instructions.txt

30 lines

```
Compilation:
1. mkdir WF
2. vi .bashrc
3. Add the line: export PATH="$PATH:
    $HOME/WF"
4. cd WF && vi cf.sh -> Write the
    compilation commands
5. mv cf.sh cf && chmod +x cf
6. Restart terminal

Kate:
1. Theme: Settings->Configure Kate->
    Color Themes
2. Vim mode: Settings->Configure Kate->
    Editing->Default input mode.
   Then Vi Input mode->Insert mode->jk =
        <esc>
3. Word wrap: Settings->Configure Kate->
    Appearance->Turn off dynamic w.w.
4. Terminal: Make sure View->Tool Views
    ->Show sidebars is on. Go to
   Settings->Configure Kate->Terminal
       and turn off Hide Konsole.
5. Hotkey for terminal: Change Focus
    Terminal Panel to F4. Click "
    Reassign"
   when it says it collides with Show
       Terminal Panel.

Fast Compile, Template, Debug:
1. cd WF && mkdir bits
2. Insert stdc++.h
3. Compile using the flags of cf.sh
4. cd .. and write template.cpp

Windows:
1. Using cmd: echo %PATH%. Using
    Powershell: echo $env:PATH
2. Add path using cmd: set PATH=%PATH%;C
    :\Program Files\CodeBlocks\MinGW\bin
   It should be the directory where g++
       is.
3. If we're using g++ of CodeBlocks,
    fsanitize won't be available :(
4. Write cf.bat at some directory.
    Ensure that directory is in PATH.
```

cf.sh

5 lines

```
#!/bin/bash

prog_name=$1

g++ "${prog_name}.cpp" -o $prog_name -
    std=c++17 -g -DDeBuG -Wall -Wshadow
    -fsanitize=address,undefined && "./
    $prog_name"
```

stdc++.h

34 lines

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

template <typename T> constexpr
void __print (const T &x);

template<typename T, typename V>
void __print(const pair<T, V> &x) {
  cerr << "{"; __print(x.first);
  cerr << ", "; __print(x.second); cerr
      << "}";
}
template <typename T> constexpr
void __print (const T &x) {
  if constexpr (is_arithmetic_v<T> ||
    is_same_v<T,const char*> ||
        is_same_v<T,bool>
    || is_same_v<T, string>) cerr << x;
  else {
    int f = 0; cerr << '{';
    for (auto &i: x)
      cerr << (f++ ? ", " : ""), __print
          (i);
    cerr << "}";
  }
}
void _print() { cerr << "]\n"; }
template <typename T, typename... V>
void _print(T t, V... v) {
  __print(t);
  if (sizeof...(v)) cerr << ", ";
  _print(v...);
}

#ifdef DeBuG
#define dbg(x...) cerr << "\t\e[93m"<<
    __func__<<":"<<__LINE__<<" [" << #x
    << "] = ["; _print(x); cerr << "\e[0
    m";
#endif
```

template.cpp

19 lines

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

#ifndef DeBuG
  #define dbg(...)
#endif

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

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0); cout.tie(0);
}
```

cf.bat

5 lines

```
@echo off
setlocal
set prog=%1
g++ %prog%.cpp -o %prog% -DDeBuG -std=c
    ++17 -g -Wall -Wshadow && .\%prog%
endlocal
```

hash.sh

3 lines

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

stress.sh

21 lines

```
#!/bin/bash

# prog_A and prog_B are the executables
    to compare

prog_A=$1
prog_B=$2
generator=$3

inp_file="inp_${generator}.txt"
out_file1="outA_${generator}.txt"
out_file2="outB_${generator}.txt"

for ((i = 1; ; ++i)) do
   echo $i
   "./$generator" > $inp_file
   "./$prog_A" < $inp_file > $out_file1
   "./$prog_B" < $inp_file > $out_file2
   diff -w "${out_file1}" "${out_file2}"
        || break
done

notify-send "bug found!!!!"
```

Mathematics (2)

2.1 Equations

$$ax^2+bx+c=0 \Rightarrow x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

The extremum is given by $x=-b/2a$.

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

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 $a_n=c_1a_{n-1}+\cdots+c_ka_{n-k}$, and $r_1,\ldots,r_k$ are distinct roots of $x^k-c_1x^{k-1}-\cdots-c_k$, there are $d_1,\ldots,d_k$ s.t.

$$a_n=d_1r_1^n+\cdots+d_kr_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n=(d_1n+d_2)r^n$.

2.3 Trigonometry

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

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

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

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

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

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

2.4 Geometry

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

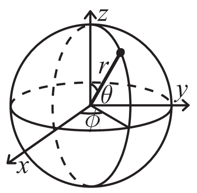
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 d/dx, $\int$

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

2.6 Sums

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

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

2.7 Series

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

Data structures (3)

OrderStatisticTree.h

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

782797, 16 lines

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

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

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

HashMap.h

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

d77092, 7 lines

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

SegmentTree.h

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

0f4bdb, 19 lines

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

segtree.cpp

deb606, 92 lines

```
template<class S> struct segtree {
  int n; vector<S> t;
  void init(int _) { n = _; t.assign(n+n
      -1, S()); }
  void init(const vector<S>& v) {
    n = sz(v); t.assign(n + n - 1, S());
    build(0,0,n-1,v);
  } template <typename... T>
  void upd(int l, int r, const T&... v)
      {
    assert(0 <= l && l <= r && r < n);
    upd(0, 0, n-1, l, r, v...);
  }
  S get(int l, int r) {
    assert(0 <= l && l <= r && r < n);
    return get(0, 0, n-1, l, r);
  }
private:
  inline void push(int u, int b, int e)
      {
    if (t[u].lazy == 0) return;
    int mid = (b+e)>>1, rc = u+((mid-b
        +1)<<1);
    t[u+1].upd(b, mid, t[u].lazy);
    t[rc].upd(mid+1, e, t[u].lazy);
    t[u].lazy = 0;
  }
  void build(int u, int b, int e, const
      vector<S>& v) {
    if (b == e) return void(t[u] = v[b])
        ;
    int mid = (b+e)>>1, rc = u+((mid-b
        +1)<<1);
    build(u+1, b, mid, v); build(rc, mid
        +1, e, v);
    t[u] = t[u+1] + t[rc];
  } template<typename... T>
  void upd(int u, int b, int e, int l,
      int r, const T&... v) {
    if (l <= b && e <= r) return t[u].
        upd(b, e, v...);
    push(u, b, e);
    int mid = (b+e)>>1, rc = u+((mid-b
        +1)<<1);
    if (l <= mid) upd(u+1, b, mid, l, r,
        v...);
    if (mid < r) upd(rc, mid+1, e, l, r,
        v...);
    t[u] = t[u+1] + t[rc];
  }
  S get(int u, int b, int e, int l, int
      r) {
    if (l <= b && e <= r) return t[u];
    push(u, b, e);
    S res; int mid = (b+e)>>1, rc = u+((
        mid-b+1)<<1);
    if (r <= mid) res = get(u+1, b, mid,
        l, r);
    else if (mid < l) res = get(rc, mid
        +1, e, l, r);
    else res = get(u+1, b, mid, l, r) +
        get(rc, mid+1, e, l, r);
    t[u] = t[u+1] + t[rc]; return res;
  }
};

/* Segment Tree
===============
Inspiration: tourist, atcoder library

(1) Declaration:
  Create a node class (sample below).
  node class must have the following:

  * A constructor (to create empty nodes
      and also to make inplace nodes).
  * + operator: returns a node which
      contains the merged information of
      two nodes.
  * upd(b, e, ...): updates this node
      representing the range [b, e]
      using information from ...

  Now, segtree<node> T; declares the
      tree.
  You can use T.init(100) to create an
      empty tree of 100 nodes in [0,
      100) range.
  You can also make a vector<node> v;
      Then put values in the vector v
      and make the tree using
   v by, T.init(v); This works in linear
      time and is faster than updating
      each individually.

(2) Usage:
  (2.1) init(int siz) or init(vector):
    Described above

  (2.2) upd(l, r, ...v):
    Update the range [l, r] with the
        information in ...
    Make sure the number of elements and
        the order of them you put here
        is the exact same
    as you declared in your node.upd()
        function.
*/

struct node {
  ll sum;
  ll lazy;

  node(ll _a = 0, ll _b = 0) : sum(_a),
      lazy(_b) {}

  node operator+(const node &obj) {
    return {sum + obj.sum, 0};
  }

  void upd(int b, int e, ll x) {
    sum += (e - b + 1) * x;
    lazy += x;
  }
};
```

UnionFindRollback.h

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

de4ad0, 21 lines

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

LineContainer.h

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

8ec1c7, 30 lines

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

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

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

Treap.h

Description: 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));
}
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

c9b7b0, 17 lines

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

PolyRoots.h

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

"Polynomial.h" b00bfe, 23 lines

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

PolyInterpolate.h

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

08bf48, 13 lines

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

BerlekampMassey.h

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

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

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

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

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

LinearRecurrence.h

Description: Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0\ldots \geq n-1]$ and $tr[0\ldots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`
Time: $\mathcal{O}\left(n^2 \log k\right)$

f4e444, 26 lines

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

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

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

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

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

4.2 Optimization

Simplex.h

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

aa8530, 68 lines

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

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

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

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

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

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

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

4.3 Matrices

Determinant.h

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

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;
}
```

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

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;
}
```

4.4 Fourier transforms

FastFourierTransform.h

Description: fft(a) computes $\hat{f}(k) = \sum_x a[x]\exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs). Otherwise, use NTT/FFTMod.
Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ (~1s for $N = 2^{22}$)

00ced6, 35 lines

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

FastFourierTransformMod.h

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

"FastFourierTransform.h" b82773, 22 lines

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

NumberTheoreticTransform.h

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

"../number-theory/ModPow.h" ced03d, 33 lines

```
const ll mod = (119 << 23) + 1, root =
     62; // = 998244353
// For p < 2^30 there is also e.g. 5 <<
     25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last
     two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
  int n = sz(a), L = 31 - __builtin_clz(
      n);
  static vl rt(2, 1);
  for (static int k = 2, s = 2; k < n; k
       *= 2, s++) {
    rt.resize(n);
    ll z[] = {1, modpow(root, mod >> s)}
        ;
    rep(i,k,2*k) rt[i] = rt[i / 2] * z[i
         & 1] % mod;
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i &
       1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i],
      a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k)
       rep(j,0,k) {
      ll z = rt[j + k] * a[i + j + k] %
          mod, &ai = a[i + j];
      a[i + j + k] = ai - z + (z > ai ?
          mod : 0);
      ai += (ai + z >= mod ? z - mod : z
          );
    }
}
vl conv(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  int s = sz(a) + sz(b) - 1, B = 32 -
      __builtin_clz(s), n = 1 << B;
  int inv = modpow(n, mod - 2);
  vl L(a), R(b), out(n);
  L.resize(n), R.resize(n);
  ntt(L), ntt(R);
  rep(i,0,n) out[-i & (n - 1)] = (ll)L[i
      ] * R[i] % mod * inv % mod;
  ntt(out);
  return {out.begin(), out.begin() + s};
}
```

FastSubsetTransform.h

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

464cf3, 16 lines

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

gcd-conv.h

Description: Computes $c_1, ..., c_n$, where $c_k = \sum_{\gcd(i,j)=k} a_i b_j$. Generate all primes upto n into pr first using sieve.
Time: $\mathcal{O}(N \log\log N)$

4769c2, 23 lines

```
void fw_multiple_transform (V<ll> &a) {
  int n = sz(a) - 1;
  for (const auto p : pr) {
    if (p > n) break;
    for (int i = n / p; i > 0; --i)
      a[i] += a[i * p];
} } // A[i] = \sum_{j} a[i * j]

void bw_multiple_transform (V<ll> &a) {
  int n = sz(a) - 1;
  for (const auto p : pr) {
    if (p > n) break;
    for (int i = 1; i * p <= n; ++i)
      a[i] -= a[i * p];
} } // From A get a

V<ll> gcd_conv (const V<ll> &a, const V<
    ll> &b) {
  assert(sz(a) == sz(b)); int n = sz(a);
  auto A = a, B = b;
  fw_multiple_transform(A);
      fw_multiple_transform(B);
  for (int i = 1; i < n; ++i) A[i] *= B[
      i];
  bw_multiple_transform(A); return A;
}
```

lcm-conv.h

Description: Computes $c_1, ..., c_n$, where $c_k = \sum_{lcm(i,j)=k} a_i b_j$. Generate all primes upto n into pr first using sieve.
Time: $\mathcal{O}(N \log\log N)$

f62031, 23 lines

```
void fw_divisor_transform (V<ll> &a) {
  int n = sz(a) - 1;
  for (const auto p : pr) {
    if (p > n) break;
    for (int i = 1; i * p <= n; ++i)
      a[i * p] += a[i];
} } // A[i] = \sum_{d | i} a[d]

void bw_divisor_transform (V<ll> &a) {
  int n = sz(a) - 1;
  for (const auto p : pr) {
    if (p > n) break;
    for (int i = n / p; i > 0; --i)
      a[i * p] -= a[i];
} } // From A get a

V<ll> lcm_conv (const V<ll> &a, const V<
    ll> &b) {
  assert(sz(a) == sz(b)); int n = sz(a);
  auto A = a, B = b;
  fw_divisor_transform(A);
      fw_divisor_transform(B);
  for (int i = 1; i < n; ++i) A[i] *= B[
      i];
  bw_divisor_transform(A); return A;
}
```

Number theory (5)

5.1 Modular arithmetic

ModInverse.h

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

6f684f, 3 lines

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

ModPow.h

b83e45, 8 lines

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

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

ModLog.h

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

c040b8, 11 lines

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

ModSum.h

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

5c5bc5, 16 lines

```
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 $0 \le a, b \le c \le 7.2 \cdot 10^{18}$.
Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow

bbbd8f, 11 lines

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

ModSqrt.h

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

"ModPow.h" 19a793, 24 lines

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

5.2 Primality

FastEratosthenes.h

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

6b2912, 20 lines

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

MillerRabin.h

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

"ModMulLL.h" 60dcd1, 12 lines

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

Factor.h

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

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

```
ull pollard(ull n) {
  auto f = [n](ull x) { return modmul(x,
      x, n) + 1; };
  ull x = 0, y = 0, t = 30, prd = 2, i =
      1, q;
  while (t++ % 40 || __gcd(prd, n) == 1)
      {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x,y) - min(
        x,y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

5.3 Divisibility

euclid.h

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

33ba8f, 5 lines

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

CRT.h

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

"euclid.h" 04d93a, 7 lines

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

5.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

phiFunction.h

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

cf7d6d, 8 lines

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

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

5.4 Fractions

5.5 Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

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

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Cycles

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

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

6.1.2 Derangements

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

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

6.1.3 Burnside's lemma

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

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

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

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

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

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.

$$p(0) = 1$$
$$p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$
$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

6.2.2 Lucas' Theorem

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

6.2.3 Binomials

multinomial.h

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

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:

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

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. $\pi(j) \ge j$, 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

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

$$C_0 = 1$$

$$C_{n+1} = \frac{2(2n+1)}{n+2}C_n$$

$$C_{n+1} = \sum C_i C_{n-i}$$

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

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

Graph (7)

7.1 Fundamentals

BellmanFord.h

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

830a8f, 23 lines

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

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

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

FloydWarshall.h

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

531245, 12 lines

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

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

66a137, 14 lines

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

7.2 Network flow

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

0ae1d4, 48 lines

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(
      n), hs(2*n), H(n) {}

  void addEdge(int s, int t, ll cap, ll
      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 addFlow(Edge& e, ll 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;
  }
  ll calc(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();
    for (Edge& e : g[s]) addFlow(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;
          for (Edge& 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)
          addFlow(*cur[u], min(ec[u],
              cur[u]->c));
        else ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a]
      >= sz(g); }
};
```

MinCostMaxFlow.h

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

fe85cc, 81 lines

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

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

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

  MCMF(int N) :
```

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

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

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

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

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

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

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

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

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.

482fe0, 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];
      for (auto 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: $\mathcal{O}\left(V^3\right)$

8b0e19, 21 lines

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

GomoryHu.h

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

"PushRelabel.h" 0418b3, 13 lines

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

7.3 Matching

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: $\mathcal{O}\left(\sqrt{V}E\right)$

f612e4, 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;
  for (int b : g[a]) if (B[b] == L + 1)
      {
    B[b] = 0;
    if (btoa[b] == -1 || dfs(btoa[b], L
        + 1, g, btoa, A, B))
      return btoa[b] = a, 1;
  }
  return 0;
}

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();
    for (int 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();
      for (int a : cur) for (int 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;
      for (int 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: $\mathcal{O}(VE)$

522b98, 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];
  for (int 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);
    for (int 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" da4196, 20 lines

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

WeightedMatching.h

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

1e0fe9, 31 lines

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

GeneralMatching.h

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

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

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

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

  if (M != N) do {
    mat.resize(M, vector<ll>(M));
    rep(i,0,N) {
      mat[i].resize(M);
      rep(j,N,M) {
        int r = rand() % mod;
        mat[i][j] = r, mat[j][i] = (mod
            - r) % mod;
      }
    }
  } while (matInv(A = mat) != M);

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

7.4 DFS algorithms

SCC.h

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

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

76b5c9, 24 lines

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

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

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.
Usage: int eid = 0; ed.resize(N);
for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) {...});
Time: $\mathcal{O}(E+V)$

2965e5, 33 lines

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

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

2sat.h

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

5f9706, 56 lines

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

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

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

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

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

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

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

EulerWalk.h

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

780b64, 15 lines

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

7.5 Coloring

EdgeColoring.h

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

e210e2, 31 lines

```
vi edgeColoring(int N, vector<pii> eds)
    {
  vi cc(N + 1), ret(sz(eds)), fan(N),
      free(N), loc;
  for (pii e : eds) ++cc[e.first], ++cc[
      e.second];
  int u, v, ncols = *max_element(all(cc)
      ) + 1;
  vector<vi> adj(N, vi(ncols, -1));
  for (pii e : eds) {
    tie(u, v) = e;
    fan[0] = v;
    loc.assign(ncols, 0);
    int at = u, end = u, d, c = free[u],
        ind = 0, i = 0;
    while (d = free[v], !loc[d] && (v =
        adj[u][d]) != -1)
      loc[d] = ++ind, cc[ind] = d, fan[
          ind] = v;
    cc[loc[d]] = c;
    for (int cd = d; at != -1; cd ^= c ^
         d, at = adj[at][cd])
      swap(adj[at][cd], adj[end = at][cd
          ^ c ^ d]);
    while (adj[fan[i]][d] != -1) {
      int left = fan[i], right = fan[++i
          ], e = cc[i];
      adj[u][e] = left;
      adj[left][e] = u;
      adj[right][e] = -1;
      free[right] = e;
    }
    adj[u][d] = fan[i];
    adj[fan[i]][d] = u;
    for (int y : {fan[0], u, end})
      for (int& z = free[y] = 0; adj[y][
          z] != -1; z++);
  }
  rep(i,0,sz(eds))
    for (tie(u, v) = eds[i]; adj[u][ret[
        i]] != v;) ++ret[i];
  return ret;
}
```

7.6 Trees

CompressTree.h

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

"LCA.h" 9775a0, 21 lines

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

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.
Time: $\mathcal{O}\left((\log N)^2\right)$

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

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

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

5909e2, 90 lines

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

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

  void link(int u, int v) { // add an
      edge (u, v)
    assert(!connected(u, v));
    makeRoot(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an
      edge (u, v)
    Node *x = &node[u], *top = &node[v];
    makeRoot(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) { // are
      u, v in the same tree?
    Node* nu = access(&node[u])->first()
        ;
    return nu == access(&node[v])->first
        ();
  }
  void makeRoot(Node* u) {
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
  Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp =
            pp; }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
};
```

DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.
Time: $\mathcal{O}(E \log V)$

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

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

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

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

7.7 Math

7.7.1 Number of Spanning Trees

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

7.7.2 Erdős–Gallai theorem

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

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

Geometry (8)

8.1 Geometric primitives

Point.h

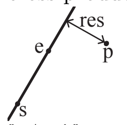
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

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.

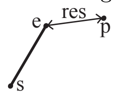


"Point.h" f6bf6b, 4 lines

```
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.



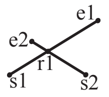
Usage: Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;

"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

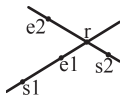
Description:
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line 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

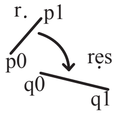
Description: Returns where *p* is as seen from *s* towards *e*. 1/0/-1 ⇔ left/on line/right. If the optional argument *eps* is given 0 is returned if *p* is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so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

CirclePolygonIntersection.h

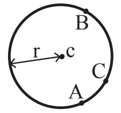
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" a1ee63, 19 lines

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

circumcircle.h

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



"Point.h" 1caa3a, 9 lines

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

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

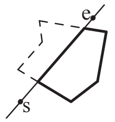
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"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.



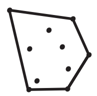
Usage: vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));

"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" 310954, 13 lines

```
typedef Point<ll> P;
```

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

HullDiameter.h

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

"Point.h" c571b8, 12 lines

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

PointInsideHull.h

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

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

```
typedef Point<ll> P;

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

LineHullIntersection.h

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

"Point.h" 7cf45b, 39 lines

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

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

8.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h" ac41a6, 17 lines

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

kdTree.h

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

"Point.h" bac5b0, 63 lines

```
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

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

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

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

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

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

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

    Node *f = node->first, *s = node->
        second;
    T bfirst = f->distance(p), bsec = s
        ->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst
        ), swap(f, s);

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

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

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" eefdf5, 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 {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p
    in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2
          ;
  return p.cross(a,b)*C + p.cross(b,c)*A
       + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new
      Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->
      o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap
      (a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

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

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

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

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

Voronoi.h

Description: Not so fast Voronoi from FastDelaunay Assumes that there are no duplicate points and that not all points are on a single line. Each circumcircle contains none of the input points. Should work for doubles as well, but haven't checked This can be optimized to use much less memory if needed. Also manually fix BIG.
Time: $\mathcal{O}(n \log n)$

"FastDelaunay.h" a1e388, 66 lines

```
struct voronoi_graph {
 using P = Point<ll>;
 using Pd = Point<double>;

 const double BIG = 1e8;

 static Pd promote (P p) { return Pd(p.x
     , p.y); }

 vector<tuple<P,P,P>> nodes;
 vector<vector<tuple<pair<P,P>, int, Pd
     >>> adj;
  // ((A, B), v, Direction)
 // the edge, when extended to a line,
     is the perpendicular bisector of
     the segment AB
 // v is the index of the adjacent node
     . it is -1 if the edge goes to
     infty
 // circumcenter(node) + Direction
     gives us the other vertex

voronoi_graph (const vector<P>& pts) {
 auto t = delaunay::triangulate(pts);
 assert (sz(t) % 3 == 0);
 nodes.resize(sz(t) / 3);
 for (int i = 0; i < sz(t); i += 3)
  nodes[i / 3] = {t[i], t[i+1], t[i+2]}
      ;
 sort(all(nodes));
 adj.resize(sz(nodes));

 vector<pair<pair<P,P>, tuple<P,P,P>>>
     delaunay_edges;
 delaunay_edges.reserve(sz(t));
 for (int i = 0; i < sz(t); i += 3) {
  for (int j = i; j < i + 3; j++)
   for (int k = j + 1; k < i + 3; k++)
    delaunay_edges.emplace_back(pair(
        min(t[j], t[k]), max(t[j], t[k
        ])), tuple(t[i], t[i+1], t[i
        +2]));
 }
 sort(all(delaunay_edges));

 for (int i = 0; i < sz(delaunay_edges)
     ; i++) {
  const int x = lower_bound(all(nodes),
      delaunay_edges[i].second) -
     nodes.begin();
  auto [a,b,c] = delaunay_edges[i].
     second;
  if (c == delaunay_edges[i].first.
     first || c == delaunay_edges[i].
     first.second)
   swap(b, c);
  if (c == delaunay_edges[i].first.
     first || c == delaunay_edges[i].
     first.second)
   swap(a, c);
  if (c == delaunay_edges[i].first.
     first || c == delaunay_edges[i].
     first.second)
   assert (false);

  if (i+1 < sz(delaunay_edges) &&
     delaunay_edges[i+1].first ==
     delaunay_edges[i].first) {
   const int y = lower_bound(all(nodes)
       , delaunay_edges[i+1].second) -
       nodes.begin();
    auto dir = get_vertex(y) -
        get_vertex(x);
    adj[x].emplace_back(delaunay_edges[i
        ].first, y, dir);
    adj[y].emplace_back(delaunay_edges[i
        ].first, x, dir * (-1.0));
   } else if (i == 0 || delaunay_edges[i
       -1].first != delaunay_edges[i].
       first) {
    bool out = (a - c).dot(b - c) < 0;
    auto dir = ((promote(a + b) / 2.0) -
        get_vertex(x)) * (out ? -1.0 :
        1.0);
    adj[x].emplace_back(delaunay_edges[i
        ].first, -1, dir * BIG);
   }
  }
 }

 Pd get_vertex (int i) {
  auto [a, b, c] = nodes[i];
  return ccCenter(promote(a), promote(b)
     , promote(c));
 }

 pair<Pd,Pd> get_edge (int i, int j) {
  const Pd vi = get_vertex(i);
  return {vi, vi + get<2>(adj[i][j])};
 }
};
```

hplane-cpalg.h

Description: Half plane intersection in O(n log n). The direction of the plane is ccw of pq vector in Halfplane struct. Usage: Status:

2e310c, 75 lines

```
const long double eps = 1e-9, inf = 1e9;

struct Point {
    long double x, y;
    explicit Point(long double x = 0,
        long double y = 0) : x(x), y(y)
        {}
    friend Point operator+(const Point &
        p, const Point &q) { return
        Point(p.x + q.x, p.y + q.y); }
    friend Point operator-(const Point &
        p, const Point &q) { return
        Point(p.x - q.x, p.y - q.y); }
    friend Point operator*(const Point &
        p, const long double &k) {
        return Point(p.x * k, p.y * k);
        }
    friend long double dot(const Point &
        p, const Point &q) { return p.x
        * q.x + p.y * q.y; }
    friend long double cross(const Point
        &p, const Point &q) { return p.
        x * q.y - p.y * q.x; }
};

struct Halfplane {
    Point p, pq;
    long double angle;
    Halfplane() {}
    Halfplane(const Point &a, const
        Point &b) : p(a), pq(b - a) {
        angle = atan2l(pq.y, pq.x);
    }
    bool out(const Point &r) { return
        cross(pq, r - p) < -eps; }
    bool operator<(const Halfplane &e)
        const { return angle < e.angle;
        }
    friend Point inter(const Halfplane &
        s, const Halfplane &t) {
        long double alpha = cross((t.p -
            s.p), t.pq) / cross(s.pq, t
            .pq);
        return s.p + (s.pq * alpha);
    }
};

vector<Point> hp_intersect(vector<
    Halfplane> &H) {

    Point box[4] = {Point(inf, inf),
        Point(-inf, inf), Point(-inf, -
        inf),
                    Point(inf, -inf)};

    for (int i = 0; i < 4; i++) {
        Halfplane aux(box[i], box[(i +
            1) % 4]);
        H.push_back(aux);
    }

    sort(H.begin(), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for (int i = 0; i < int(H.size()); i
        ++) {
        while (len > 1 && H[i].out(inter
            (dq[len - 1], dq[len - 2])))
             {
            dq.pop_back(); --len;
        }
        while (len > 1 && H[i].out(inter
            (dq[0], dq[1]))) {
            dq.pop_front(); --len;
        }
        if (len > 0 && fabsl(cross(H[i].
            pq, dq[len - 1].pq)) < eps)
            {
            if (dot(H[i].pq, dq[len -
                1].pq) < 0.0)
                return vector<Point>();
            if (H[i].out(dq[len - 1].p))
                 {
                dq.pop_back();
                --len;
            } else
                continue;
        }
        dq.push_back(H[i]);
        ++len;
    }
    while (len > 2 && dq[0].out(inter(dq
        [len - 1], dq[len - 2]))) {
        dq.pop_back(); --len;
    }

    while (len > 2 && dq[len - 1].out(
        inter(dq[0], dq[1]))) {
        dq.pop_front(); --len;
    }
    if (len < 3)
        return vector<Point>();
    vector<Point> ret(len);
    for (int i = 0; i + 1 < len; i++) {
        ret[i] = inter(dq[i], dq[i + 1])
            ;
    }
    ret.back() = inter(dq[len - 1], dq
        [0]);
    return ret;
}
```

8.5 3D

PolyhedronVolume.h

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

3058c3, 6 lines

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

Point3D.h

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

8058ae, 32 lines

```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0)
      : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y,
        p.z); }
  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" 5b45fc, 49 lines

```
typedef Point3D<double> P3;

struct PR {
  void ins(int x) { (a == -1 ? a : b) =
      x; }
  void rem(int x) { (a == x ? a : b) =
      -1; }
  int cnt() { return (a != -1) + (b !=
      -1); }
  int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR
      >(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int
       l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A
        [i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c)
        .ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])
          ) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() !=
    2) mf(f.a, f.b, i, f.c);
      C(a, b, c); C(a, c, b); C(b, c, a)
          ;
    }
  }
  for (F& it : FS) if ((A[it.b] - A[it.a
      ]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0)
        swap(it.c, it.b);
  return FS;
};
```

sphericalDistance.h

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

611f07, 8 lines

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

ee09e2, 12 lines

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

Manacher.h

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

e7ad79, 13 lines

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

MinRotation.h

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

d07a42, 8 lines

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

SuffixArray.h

Description: Builds suffix array for a string. `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;
  }
};
```

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 Dynamic programming

KnuthDP.h

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

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

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \le k < hi(i)}(f(i, k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.

Time: $\mathcal{O}((N + (hi - lo)) \log N)$

d38d2b, 18 lines

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

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

10.3 Optimization tricks

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

10.3.1 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.

FastMod.h

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

751a02, 8 lines

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

Arman (11)

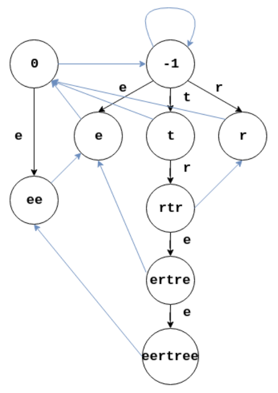
bridges-and-points.cpp

Description: Only need to call `PointsAndBridges()`. Nodes are $[0, n)$ which can easily be configured there.
Time: $\mathcal{O}(V+E)$ except the final sorting of bridges. If the graph doesn't contain any multi-edges, that part can be omitted.

a8990e, 40 lines

```
vector<bool> vis, cutPoint;
vi low, disc; int tim;
vector<pair<int,int>> mebi, bridge;

void dfsPB(int u, int f = -1) {
  vis[u] = true; int children = 0;
  disc[u] = low[u] = tim++;
  for (int v : g[u]) {
    if (v == f) continue; // all loops
        ignored
    if (vis[v]) low[u] = min(low[u],
        disc[v]);
    else {
      dfsPB(v, u); ++children;
      low[u] = min(low[u], low[v]);

      if (disc[u] < low[v]) {
        // u === v if no multi edges.
        mebi.pb({min(u, v), max(u, v)});
      }
      if (disc[u] <= low[v]  && f != -1)
        cutPoint[u] = true;//this line
            executes > once
    }
  }
  if (f == -1 && children > 1) cutPoint[
      u] = 1;
}

void PointsAndBridges() { // [0,n)
  vis.assign(n, false); tim = 0;
  low.assign(n, -1); disc.assign(n, -1);
  cutPoint.assign(n, false); mebi.clear
      ();

  for (int i = 0; i < n; ++i)
    if (!vis[i]) dfsPB(i);

  sort(all(mebi)); bridge.clear();
  for (int i = 0; i < sz(mebi); ++i) {
    if ((i + 1 < sz(mebi) && mebi[i + 1]
        == mebi[i])
      || (i > 0 && mebi[i - 1] == mebi[i
          ])) continue;
    bridge.pb(mebi[i]);
  }
}
```

11.1 Palindromic Tree



palindromic-tree.cpp

Description: Makes a trie of $\mathcal{O}(|S|)$ vertices containing all distinct palindromes of a string. Suffix links give the longest proper suffix/prefix of that palindrome which is also a palindrome.
Usage: S := 1-indexed string. {add} characters left to right.
After adding the i-th character {ptr} points to the node containing the longest palindrome ending at i.
Time: $\mathcal{O}(|S|)$

08d4ad, 41 lines

```
const int ALPHA = 26;
struct PalindromicTree {
  struct node {
    int to[ALPHA];
    int link, len;

    node(int a = 0, int b = 0) : link(a)
        , len(b) {
      memset(to, 0, sizeof to);
    }
  };

  vector<node> T; int ptr;

  int ID(char x) { return x - 'a'; }
  void init() {
    T.clear(); ptr = 1;
    T.emplace_back(0, -1); // Odd root
    T.emplace_back(0, 0);  // Even root
  }

  void append(int i, string &s) {
    while (s[i - T[ptr].len - 1] != s[i
        ])
      ptr = T[ptr].link;
    int id = ID(s[i]);
    // if node already exists, return
    if (T[ptr].to[id]) return void(ptr =
         T[ptr].to[id]);
    int tmp = T[ptr].link;
    while (s[i - T[tmp].len - 1] != s[i
        ])
      tmp = T[tmp].link;

    int newlink = T[ptr].len == -1 ? 1 :
         T[tmp].to[id];

    // ptr is the parent of this new
        node
    T.emplace_back(newlink, T[ptr].len +
         2);

    // Now shift ptr to the newly
        created node
    T[ptr].to[id] = sz(T) - 1;
    ptr = sz(T) - 1;
  }
};
```

11.2 Aho Corasick

ahoCorasick.h

Usage: insert strings first (0-indexed). Then call prepare to use everything. link = suffix link. to[ch] = trie transition. jump[ch] = aho transition to ch using links.
Time: $\mathcal{O}(AL)$

36fabb, 37 lines

```
const int L = 5000; // Total no of
      characters
const int A = 10; // Alphabet size

struct Aho_Corasick {
  struct Node {
    bool end_flag; int par, pch, to[A],
        link, jump[A];
    Node() {
      par = link = end_flag = 0;
      memset(to, 0, sizeof to);
      memset(jump, 0, sizeof jump);
    }
  };
  Node t[L]; int at;
  Aho_Corasick() { at = 0; }

  void insert(string &s) {
    int u = 0;
    for (auto ch : s) {
      int &v = t[u].to[ch - '0'];
      if (!v) v = ++at;
      t[v].par = u; t[v].pch = ch - '0';
           u = v;
    }
    t[u].end_flag = true;
  }

  void prepare() {
    for (queue<int> q({0}); !q.empty();
        q.pop()) {
      int u = q.front(), w = t[u].link;
      for (int ch = 0; ch < A; ++ch) {
        int v = t[u].to[ch];
        if (v) {
          t[v].link = t[w].jump[ch];
          q.push(v);
        }
        t[u].jump[ch] = v ? v : t[w].
             jump[ch];
  } } }
}aho;
```

11.3 Sparse Table

sparsetable.cpp

Description: 0-Indexed, Query type $[l, r)$. Handles range query on static arrays.
Usage: `SparseTable<int, op> table;`
Time: $\mathcal{O}(n \lg n)$ to construct. query is $\mathcal{O}(1)$ if function is idempotent ($f \circ f = f$). Otherwise, use `lgQuery`, which is $\mathcal{O}(\lg n)$.

40bbc0, 23 lines

```
template<typename T, T (*op)(T, T)>
struct SparseTable {
  vector<vector<T>> t;
  SparseTable(const vector<T> &v) : t(1,
        v) {
    for (int j = 1; j <= __lg(sz(v)); ++
        j) {
      t.emplace_back(sz(v) - (1 << j) +
          1);
      for (int i = 0; i < sz(t[j]); ++i)
        t[j][i] = op(t[j - 1][i],
          t[j - 1][i + (1 << (j - 1))]);
  } }
  T query(int l, int r) { assert(l < r);
    int k = __lg(r - l);
    return op(t[k][l], t[k][r - (1 << k)
        ]);
  }
  T lgQuery(int l, int r) { assert(l < r
      );
    T ret = t[0][l++]; if (l == r)
      return ret;
    for (int j = __lg(r - l); j >= 0; --
        j) {
      if (l + (1 << j) - 1 < r) {
        ret = op(ret, t[j][l]);
        l += (1 << j);
    } } return ret;
  }
}; int op(int a, int b) { return min(a,
    b); }
```

11.4 Tree Binarize

treebinarize.h

Description: Given weighted graph `g` with nodes $\in [1, n]$, makes a new binary tree `T` with nodes $\in$ [1,`nnode`) such that distance is maintained. Adds at-most $2(N-1)$ nodes (actually much less than that).g must have (w, v) pairs.

46dd6e, 23 lines

```
struct BinaryTree {
  int nnode;
  V<V<pii>> T;
  void dfs(int u, int f) {
    for (auto &e : T[u])
      e.second == f ? swap(e, T[u][0]) :
           dfs(e.second, u);
  }
  BinaryTree(V<V<pii>> &g, int I = 1) :
      T(g) {
    dfs(I, -1); int n = sz(T);
    for (int u = I; u < n; ++u) {
      for (int i = 2 - (u == I), x = u;
          i+1 < sz(T[u]); ++i) {
        T.push_back({{0, x}, T[u][i], T[
            u][i+1]});
        int v1 = T[u][i].second, v2 = T[
            u][i+1].second;
        T[v1][0] = T[v2][0] = {1, sz(T)
            - 1};
        T[x][2 - (x == I)] = {0, sz(T) -
            1};
        x = sz(T) - 1;
      }
      if (sz(T[u]) > 3 - (u == I))
        T[u].resize(3 - (u == I));
    }
    nnode = sz(T) - 1;
  }
};
```

11.5 Centroid Decomposition

centroidDecomp.cpp

Description: Builds the Centroid Tree of the tree `adj`. For each centroid `c`, calculates its parent `C[c].p`, all outgoing children in `C[c].out` and the (index of `C[parent of c].out` which points to `c` itself) in `C[c].p_idx`. Just call `build()`. Parent of `ROOT` = -1.
Time: `build()` in $\mathcal{O}(n \lg n)$.

34b647, 35 lines

```
struct centroidDecomp {
  struct centroid {
    int p, p_idx; vi out;
    centroid() { p = p_idx = -1; };
  };
  int ROOT; vector<centroid> C;
  vector<bool> done; vi siz;
  void build() {
    C.resize(sz(adj)); done.resize(sz(
        adj), false);
    siz.resize(sz(adj)); ROOT =
        build_tree(1, -1);
  }
  int dfs(int u, int f) {
    siz[u] = 1;
    for (int v : adj[u]) if (v != f && !
        done[v])
        siz[u] += dfs(v, u);
    return siz[u];
  }
  int find_centroid(int u, int f, int
      lim) {
    for (int v : adj[u])
      if (v != f && !done[v] && 2*siz[v]
          > lim)
        return find_centroid(v, u, lim);
    return u;
  }
  int build_tree(int u, int f, int lev =
       0) {
    dfs(u, f); if (siz[u] == 1) return u
        ;
    int c = find_centroid(u, f, siz[u]);
    done[c] = true;
    for (int v : adj[c]) if (!done[v]) {
      int next_c = build_tree(v, c);
      // next_c is the next centroid
          after c.
      C[next_c].p = c;
      C[next_c].p_idx = sz(C[c].out);
      C[c].out.pb(next_c);
    } return c;
} }cd;
```

11.6 Functional Graph

functionalGraph.h

Description: Functional graph essentials. $f : [0, n) \to [0, n)$. lev: distance from entering cycle, 0 if on cycle. pos: gives an ordering of nodes on same cycle. clen: no. of nodes on the cycle containing u, -1 if not on one. dsu: merges all edges as bidirectional. sub: merges all but cycle edges, parents are on cycle.
Time: Linear

9031d6, 57 lines

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

struct fGraph {
  int n;
  V<int> f, lev, pos, clen;
  DSU dsu, sub;

  fGraph (const V<int> &ff) : n(sz(ff)),
       f(ff),
  lev(n, 0), dsu(n), sub(n) {
    for (int i = 0; i < n; ++i) lev[f[i
        ]]++;
    queue<int> q; stack<int> rev;
    for (int i = 0; i < n; ++i)
      if (!lev[i]) q.push(i);
    while (!q.empty()) {// from leaves
        to cycle
      int u = q.front(); q.pop();
      rev.push(u);
      if (!--lev[f[u]]) q.push(f[u]);
    }
    for (int i = 0; i < n; ++i) {
      dsu.join(f[i], i);
      if (!lev[i]) sub.join(f[i], i);
      lev[i] = (lev[i] == 0 ? -1 : 0);
    }
    while (!rev.empty()) {
      int u = rev.top(); rev.pop(); //
          top to leaves
      lev[u] = lev[f[u]] + 1;
    }
    pos.assign(n, -1);
    clen.assign(n, -1);
    for (int i = 0; i < m; ++i)
      if (pos[i] == -1 && !lev[i]) {
        int len = 0; // iterates on
             cycle
        for (int u = i; pos[u] == -1; u
            = f[u])
          pos[u] = len++;
        for (int u = i; clen[u] == -1; u
             = f[u])
          clen[u] = len;
      }
  }
  bool connected (int u, int v)
    { return dsu.find(u) == dsu.find(v);
         }
  bool sameTree (int u, int v)
    { return sub.find(u) == sub.find(v);
         }
};
```