University of Tartu ICPC Team Notebook (2018-2019) March 10, 2019

1 Setup

```
set smartindent cindent
set ts=4 sw=4 expandtab
syntax enable
set clipboard=unnamedplus
# setxkbmap -option caps:escape
# valgrind --vgdb-error=0 ./a <inp &
# gdb a
# target remote | vgdb
```

2 crc.sh

```
#!/bin/envbash
for j in `seq 5 5 200`; do
  sed '/^\s*$/d' $1 | head -$j | tr -d '[[:space:]]' | cksum | cut -f1
  ↪  -d ' ' | tail -c 4 #whistespaces don't matter.
done #there shouldn't be any COMMENTS.
#copy lines being checked to separate file.
# $ ./crc.sh tmp.cpp
```

3 gcc ordered set

```
#define DEBUG(...) cerr << __VA_ARGS__ << endl;
#ifndef CDEBUG
#undef DEBUG
#define DEBUG(...) ((void)0);
#define NDEBUG@                                                    #438 @
#endif
#define ran(i, a, b) for (auto i = (a); i < (b); i++)
#include<bits/stdc++.h>
typedef long long ll;
typedef long double ld                                             #546
using namespace std;#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T                                               #822
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
  tree_order_statistics_node_update>;
intmain({
  ordered_set<int> cur;
  cur.insert(1)                                                    #325
  cur.insert(3);
  cout << cur.order_of_key(2)
      << endl;// the number of elements in the set less than 2
```

```
cout << *cur.find_by_order(0)
     << endl;// the 0-th smallest number in the set(0-based)     #478
cout << *cur.find_by_order(1)
     << endl;// the 1-th smallest number in the set(0-based)
```

4 2D geometry

Define $\mathbf{orient}(A,B,C) = \overline{AB}\times\overline{AC}$. CCW iff >0.
Define $\mathbf{perp}((a,b)) = (-b,a)$. The vectors are orthogonal.
For line $ax+by=c$ def $\overline{v} = (-b,a)$.
Line through P and Q has $\overline{v}=\overline{PQ}$ and $c=\overline{v}\times P$.
$\mathbf{side}_l(P) = \overline{v_l}\times P - c_l$ sign determines which side P is on from l.
$\mathbf{dist}_l(P) = \mathbf{side}_l(P)/\|v_l\|$ squared is integer.
Sorting points along a line: comparator is $\overline{v}\cdot A < \overline{v}\cdot B$.
Translating line by $\overline{t}$: new line has $c' = c + \overline{v}\times\overline{t}$.
Line intersection: is $(c_l\overline{v}_m - c_m\overline{v}_l)/(\overline{v}_l\times\overline{v}_m)$.
Project P onto l: is $P - \mathbf{perp}(v)\,\mathbf{side}_l(P)/\|v\|^2$.
Angle bisectors: $\overline{v} = \overline{v}_l/\|\overline{v}_l\| + \overline{v}_m/\|\overline{v}_m\|$
$c = c_l/\|\overline{v}_l\| + c_m/\|\overline{v}_m\|$.
P is on segment AB iff $\mathbf{orient}(A,B,P)=0$ and $\overline{PA}\cdot\overline{PB}\leqslant 0$.
Proper intersection of AB and CD exists iff $\mathbf{orient}(C,D,A)$ and $\mathbf{orient}(C,D,B)$ have opp. signs and $\mathbf{orient}(A,B,C)$ and $\mathbf{orient}(A,B,D)$ have opp. signs. Coordinates:

$$\frac{A\,\mathbf{orient}(C,D,B) - B\,\mathbf{orient}(C,D,A)}{\mathbf{orient}(C,D,B) - \mathbf{orient}(C,D,A)}.$$

Circumcircle center:

```
pt circumCenter(pt a, pt b, pt c) {
  b = b-a, c = c-a; // consider coordinates
      relative to A
  assert(cross(b,c) != 0); // no circumcircle if
       A,B,C aligned
  return a + perp(b*sq(c) - c*sq(b))/cross(b,c)
      /2;
```

Circle-line intersect:

```
int circleLine(pt o, double r, line l, pair<pt,
    pt> &out) {
  double h2 = r*r - l.sqDist(o);
  if (h2 >= 0) { // the line touches the circle
    pt p = l.proj(o); // point P
    pt h = l.v*sqrt(h2)/abs(l.v); // vector
        paral to l, of len h
    out = {p-h, p+h};
  }
  return 1 + sgn(h2);
```

Circle-circle intersect:

```
int circleCircle(pt o1, double r1, pt o2, double
     r2, pair<pt,pt> &out) {
  pt d=o2-o1; double d2=sq(d);
  if (d2 == 0) {assert(r1 != r2); return 0;} //
      concentric circles
  double pd = (d2 + r1*r1 - r2*r2)/2; // = |O_1P
      | * d
  double h2 = r1*r1 - pd*pd/d2; // = h^2
  if (h2 >= 0) {
    pt p = o1 + d*pd/d2, h = perp(d)*sqrt(h2/d2)
      ;
    out = {p-h, p+h};}
  return 1 + sgn(h2);
```

Tangent lines:

```
int tangents(pt o1, double r1, pt o2, double r2,
     bool inner, vector<pair<pt,pt>> &out) {
  if (inner) r2 = -r2;
  pt d = o2-o1;
  double dr = r1-r2, d2 = sq(d), h2 = d2-dr*dr;
  if (d2 == 0 || h2 < 0) {assert(h2 != 0);
      return 0;}
  for (double sign : {-1,1}) {
    pt v = (d*dr + perp(d)*sqrt(h2)*sign)/d2;
    out.push_back({o1 + v*r1, o2 + v*r2});}
  return 1 + (h2 > 0);
```

5 3D geometry

$\mathbf{orient}(P,Q,R,S) = (\overline{PQ}\times\overline{PR})\cdot\overline{PS}$.
S above PQR iff >0.
For plane $ax+by+cz=d$ def $\overline{n} = (a,b,c)$.
Line with normal $\overline{n}$ through point P has $d=\overline{n}\cdot P$.
$\mathbf{side}_\Pi(P) = \overline{n}\cdot P - d$ sign determines side from Π.
$\mathbf{dist}_\Pi(P) = \mathbf{side}_\Pi(P)/\|\overline{n}\|$.
Translating plane by $\overline{t}$ makes $d' = d + \overline{n}\cdot\overline{t}$.
Plane-plane intersection of has direction $\overline{n}_1\times\overline{n}_2$ and goes through $((d_1\overline{n}_2 - d_2\overline{n}_1)\times\overline{d})/\|\overline{d}\|^2$.
Line-line distance:

```
double dist(line3d l1, line3d l2) {
  p3 n = l1.d*l2.d;
  if (n == zero) // parallel
    return l1.dist(l2.o);
  return abs((l2.o-l1.o)|n)/abs(n);
```

Spherical to Cartesian:
$(r\cos\varphi\cos\lambda, r\cos\varphi\sin\lambda, r\sin\varphi)$.
Sphere-line intersection:

```
int sphereLine(p3 o, double r, line3d l, pair<p3
    ,p3> &out) {
  double h2 = r*r - l.sqDist(o);
  if (h2 < 0) return 0; // the line doesn't
      touch the sphere
  p3 p = l.proj(o); // point P
  p3 h = l.d*sqrt(h2)/abs(l.d); // vector
      parallel to l, of length h
  out = {p-h, p+h};
  return 1 + (h2 > 0);
```

Great-circle distance between points A and B is $r\angle AOB$.
Spherical segment intersection:

```
bool properInter(p3 a, p3 b, p3 c, p3 d, p3 &out
    ) {
  p3 ab = a*b, cd = c*d; // normals of planes
      OAB and OCD
  int oa = sgn(cd|a),
    ob = sgn(cd|b),
    oc = sgn(ab|c),
    od = sgn(ab|d);
  out = ab*cd*od; // four multiplications =>
      careful with overflow !
  return (oa != ob && oc != od && oa != oc);
}
bool onSphSegment(p3 a, p3 b, p3 p) {
  p3 n = a*b;
  if (n == zero)
    return a*p == zero && (a|p) > 0;
  return (n|p) == 0 && (n|a*p) >= 0 && (n|b*p)
      <= 0;
}
struct directionSet : vector<p3> {
  using vector::vector; // import constructors
  void insert(p3 p) {
    for (p3 q : *this) if (p*q == zero) return;
    push_back(p);
  }
};
directionSet intersSph(p3 a, p3 b, p3 c, p3 d) {
  assert(validSegment(a, b) && validSegment(c, d
      ));
  p3 out;
  if (properInter(a, b, c, d, out)) return {out
      };
  directionSet s;
  if (onSphSegment(c, d, a)) s.insert(a);
  if (onSphSegment(c, d, b)) s.insert(b);
  if (onSphSegment(a, b, c)) s.insert(c);
  if (onSphSegment(a, b, d)) s.insert(d);
  return s;
}
```

Angle between spherical segments AB and AC is angle between $A\times B$ and $A\times C$.
Oriented angle: subtract from 2π if mixed product is negative.
Area of a spherical polygon:

$$r^2[\text{sum of interior angles} - (n-2)\pi].$$

6 Triangle centers

```
const double min_delta = 1e-13;
const double coord_max = 1e6;
typedef complex<double> point;
point A, B, C;// vertixes of the triangle
boolcollinear(                                                    #823
  double min_diff = min(abs(A - B), min(abs(A - C), abs(B - C)));
  if (min_diff < coord_max * min_delta) return true;
  point sp = (B - A) / (C - A);
  double ang =
    M_PI / 2                                                      #638
    abs(abs(arg(sp)) - M_PI / 2);// positive angle with the real line
  return ang < min_delta;
                                                                  %446
pointcircum_center({
  if (collinear()) return point(NAN, NAN);
 // squared lengths of sides
  double a2 = norm(B - C);
  double b2 = norm(A - C);
  double c2 = norm(A - B)                                         #715
 // barycentric coordinates of the circumcenter
  double c_A = a2 * (b2 + c2 - a2);// sin(2 * alpha) works also
  double c_B = b2 * (a2 + c2 - b2);
  double c_C = c2 * (a2 + b2 - c2);
  double sum = c_A + c_B + c_C;
  c_A /= sum                                                      #407
  c_B /= sum;
  c_C /= sum;
  return c_A * A + c_B * B + c_C * C;// cartesian
                                                                  %856
pointcentroid({// center of mass
  return (A + B + C) / 3.0;
}
point ortho_center() {// euler line
  point O = circum_center()                                       #895
  return O + 3.0 * (centroid() - O);
};
pointnine_point_circle_center({// euler line
  point O = circum_center();
  return O + 1.5 * (centroid() - O)                               #193
}                                                                 %031
pointin_center({
  if (collinear()) return point(NAN, NAN);
  double a = abs(B - C);// side lenghts
  double b = abs(A - C);
  double c = abs(A - B)                                           #954
 // trilinear coordinates are (1,1,1)
  double sum = a + b + c;
  a /= sum;
  b /= sum;
  c /= sum;                    // barycentric
  return a * A + b * B + c * C;// cartesian                       #596
```

7 Seg-Seg intersection, halfplane intersection area

```
struct Seg {
  Vec a, b;
  Vecd({ return b - a; }
};
Vecintersection(Seg l, Seg r                                      #327
  Vec dl = l.d(), dr = r.d();
  if (cross(dl, dr) == 0) return {nanl(""), nanl("")};
  double h = cross(dr, l.a - r.a) / len(dr);
  double dh = cross(dr, dl) / len(dr);
  return l.a + dl * (h / -dh)                                     #893
}// Returns the area bounded by halfplanes
doublecalc_area(vector<Seg> lines{
  double lb = -HUGE_VAL, ub = HUGE_VAL;
  vector<Seg> linesBySide[2];
  for (auto line : lines)                                         #454
    if (line.b.y == line.a.y) {
      if (line.a.x < line.b.x) {
        lb = max(lb, line.a.y);
      } else {
        ub = min(ub, line.a.y)                                    #029
      }
    } else if (line.a.y < line.b.y) {
      linesBySide[1].push_back(line);
    } else {
      linesBySide[0].push_back({line.b, line.a})                  #613
    }
  }
  sort(
    linesBySide[0].begin(), linesBySide[0].end(), [](Seg l, Seg r) {
      if (cross(l.d(), r.d()) == 0                                #123
        returnnormal(l.d() * l.a > normal(r.d()) * r.;
      returncross(l.d(, r.d()) < ;
    });
  sort(
    linesBySide[1].begin(), linesBySide[1].end(), [](Seg l, Seg r)
    ↪ #115
      if (cross(l.d(), r.d()) == 0)
        return normal(l.d()) * l.a < normal(r.d()) * r.a;
      returncross(l.d(, r.d()) > ;
    });
 // Now find the application area of the lines and clean up redundant
 // ones
  vector<double> applyStart[2]                                    #597
  for (int side = 0; side < 2; side++) {
    vector<double> &apply = applyStart[side];
    vector<Seg> curLines;
    for (auto line : linesBySide[side]) {
      while (curLines.size() > 0)                                 #412
        Seg other = curLines.back();
```

```
      if (cross(line.d(), other.d()) != 0) {
        double start = intersection(line, other).y;
        if (start > apply.back()) break;
                                                              #503
      curLines.pop_back();
      apply.pop_back();
    }
    if (curLines.size() == 0) {
      apply.push_back(-HUGE_VAL)                              #321
    } else {
      apply.push_back(intersection(line, curLines.back()).y);
    }
    curLines.push_back(line);
                                                              #047
  linesBySide[side] = curLines;
}
applyStart[0].push_back(HUGE_VALL);
applyStart[1].push_back(HUGE_VALL);
double result = 0                                             #908
{
  double lb = -HUGE_VALL, ub;
  for (int i = 0, j = 0; i < (int)linesBySide[0].size() &&
                         j < (int)linesBySide[1].size();
     lb = ub)                                                 #251
    ub = min(applyStart[0][i + 1], applyStart[1][j + 1]);
    double alb = lb, aub = ub;
    Seg l0 = linesBySide[0][i], l1 = linesBySide[1][j];
    if (cross(l1.d(), l0.d()) > 0) {
      alb = max(alb, intersection(l0, l1).y)                  #743
    } else if (cross(l1.d(), l0.d()) < 0) {
      aub = min(aub, intersection(l0, l1).y);
    }
    alb = max(alb, lb);
    aub = min(aub, ub)                                        #839
    aub = max(aub, alb);
    {
      double x1 = l0.a.x + (alb - l0.a.y) / l0.d().y * l0.d().x;
      double x2 = l0.a.x + (aub - l0.a.y) / l0.d().y * l0.d().x;
      result -= (aub - alb) * (x1 + x2) / 2                   #075
    }
    {
      double x1 = l1.a.x + (alb - l1.a.y) / l1.d().y * l1.d().x;
      double x2 = l1.a.x + (aub - l1.a.y) / l1.d().y * l1.d().x;
      result += (aub - alb) * (x1 + x2) / 2                   #717
    }
    if (applyStart[0][i + 1] < applyStart[1][j + 1]) {
      i++;
    } else {
      j++                                                     #446
    }
  }
}
  return result;
```

8 Convex polygon algorithms

```
typedef pair<int, int> Vec;
typedef pair<Vec, Vec> Seg;
typedef vector<Seg>::iterator SegIt;#define F first
#define S second@                                                  #608 @
#define MP(x, y) make_pair(x,y)
lldot(Vec &v1, Vec &v2{ return (ll)v1.F * v2.F + (ll)v1.S * v2.S; }
llcross(Vec &v1, Vec &v2{
  return (ll)v1.F * v2.S - (ll)v2.F * v1.S;
                                                                   #541
lldist_sq(Vec &p1, Vec &p2{
  return (ll)(p2.F - p1.F) * (p2.F - p1.F) +
         (ll)(p2.S - p1.S) * (p2.S - p1.S);
                                                                   %008
struct Hull {
  vector<Seg> hull;
  SegIt up_beg;
  template <typename It>
  void extend(It beg, It end) {// O(n)                             #096
    vector<Vec> r;
    for (auto it = beg; it != end; ++it) {
      if (r.empty() || *it != r.back()) {
        while (r.size() >= 2) {
          int n = r.size()                                         #442
          Vec v1 = {r[n - 1].F - r[n - 2].F, r[n - 1].S - r[n - 2].S};
          Vec v2 = {it->F - r[n - 2].F, it->S - r[n - 2].S};
          if (cross(v1, v2) > 0) break;
          r.pop_back();
                                                                   #605
        r.push_back(*it);
      }
    }
    ran(i, 0, (int)r.size() - 1) hull.emplace_back(r[i], r[i + 1]);
                                                                   #572
  Hull(vector<Vec> &vert) {        // atleast 2 distinct points
    sort(vert.begin(), vert.end());// O(n log(n))
    extend(vert.begin(), vert.end());
    int diff = hull.size();
    extend(vert.rbegin(), vert.rend())                             #964
    up_beg = hull.begin() + diff;
                                                                   %722
  boolcontains(Vec p{// O(log(n))
    if (p < hull.front().F || p > up_beg->F) return false;
    {
      auto it_low = lower_bound(
        hull.begin(), up_beg, MP(MP(p.F, (int)-2e9), MP(0, 0)))   #542
      if (it_low != hull.begin()) --it_low;
      Vec a = {it_low->S.F - it_low->F.F, it_low->S.S - it_low->F.S};
      Vec b = {p.F - it_low->F.F, p.S - it_low->F.S};
```

```
    if (cross(a, b) < 0)// < 0 is inclusive, <=0 is exclusive
      return false                                              #681
  }
  {
    auto it_up = lower_bound(hull.rbegin(),
      hull.rbegin() + (hull.end() - up_beg),
      MP(MP(p.F, (int)2e9), MP(0, 0)))                          #423
    if (it_up - hull.rbegin() == hull.end() - up_beg) --it_up;
    Vec a = {it_up->F.F - it_up->S.F, it_up->F.S - it_up->S.S};
    Vec b = {p.F - it_up->S.F, p.S - it_up->S.S};
    if (cross(a, b) > 0)// > 0 is inclusive, >=0 is exclusive
      return false                                              #227
  }
  return true;
                                                                %826
// The function can have only one local min and max
// and may be constant only at min and max.
 template <typename T>
 SegIt max(function<T(Seg &)> f) {// O(log(n))
  auto l = hull.begin();
  auto r = hull.end();
  SegIt b = hull.end()                                          #566
  T b_v;
  while (r - l > 2) {
    auto m = l + (r - l) / 2;
    T l_v = f(*l);
    T l_n_v = f(*(l + 1))                                       #586
    T m_v = f(*m);
    T m_n_v = f(*(m + 1));
    if (b == hull.end() || l_v > b_v) {
      b = l;// If max is at l we may remove it from the range.
      b_v = l_v                                                 #332
    }
    if (l_n_v > l_v) {
      if (m_v < l_v) {
        r = m;
      } else                                                    #279
        if (m_n_v > m_v) {
          l = m + 1;
        } else {
          r = m + 1;
                                                                #656
        }
    } else {
      if (m_v < l_v) {
        l = m + 1;
      } else                                                    #311
        if (m_n_v > m_v) {
          l = m + 1;
        } else {
          r = m + 1;
                                                                #469
        }
      }
    }
    T l_v = f(*l);
    if (b == hull.end() || l_v > b_v)                           #864
      b = l;
      b_v = l_v;
    }
    if (r - l > 1) {
      T l_n_v = f(*(l + 1))                                     #972
      if (b == hull.end() || l_n_v > b_v) {
        b = l + 1;
        b_v = l_n_v;
      }
                                                                #086
    return b;
                                                                %504
SegItclosest(Vec p{// p can't be internal(can be on border),
                   // hull must have atleast 3 points
  Seg &ref_p = hull.front();// O(log(n))
  returnmax(function<double(Seg &>(
    [&p, &ref_p](
      Seg &seg){// accuracy of used type should be coord^2     #071
      if (p == seg.F) return 10 - M_PI;
      Vec v1 = {seg.S.F - seg.F.F, seg.S.S - seg.F.S};
      Vec v2 = {p.F - seg.F.F, p.S - seg.F.S};
      ll c_p = cross(v1, v2);
      if (c_p > 0) {// order the backside by angle             #868
        Vec v1 = {ref_p.F.F - p.F, ref_p.F.S - p.S};
        Vec v2 = {seg.F.F - p.F, seg.F.S - p.S};
        ll d_p = dot(v1, v2);
        ll c_p = cross(v2, v1);
        returnatan2(c_p, d_p /                                  #384
      }
      ll d_p = dot(v1, v2);
      double res = atan2(d_p, c_p);
      if (d_p <= 0 && res > 0) res = -M_PI;
      if (res > 0)                                              #050
        res += 20;
      } else {
        res = 10 - res;
      }
      return res                                                #631
    }));
                                                                %283
template <int DIRECTION>// 1 or -1
Vectan_point(Vec p{  // can't be internal or on border
 //-1 iff CCW rotation of ray from p to res takes it away from
 // polygon?
  Seg &ref_p = hull.front();// O(log(n))
```

```
  auto best_seg = max(function<double(Seg &)>(
    [&p, &ref_p]                                                  #209
      Seg &seg) {// accuracy of used type should be coord^2
      Vec v1 = {ref_p.F.F - p.F, ref_p.F.S - p.S};
      Vec v2 = {seg.F.F - p.F, seg.F.S - p.S};
      ll d_p = dot(v1, v2);
      ll c_p = DIRECTION * cross(v2, v1)                          #762
      returnatan2(c_p, d_p;// order by signed angle
    }));
  return best_seg->F;
                                                                  %037
SegItmax_in_dir(Vec v{// first is the ans. O(log(n))
  returnmax(
    function<ll(Seg &>([&v](Seg &seg){ return dot(v, seg.F); }));
                                                                  %596
pair<SegIt, SegIt> intersections(Seg l) {// O(log(n))
  int x = l.S.F - l.F.F;
  int y = l.S.S - l.F.S;
  Vec dir = {-y, x};
  auto it_max = max_in_dir(dir)                                   #740
  auto it_min = max_in_dir(MP(y, -x));
  ll opt_val = dot(dir, l.F);
  if (dot(dir, it_max->F) < opt_val ||
      dot(dir, it_min->F) > opt_val)
    return MP(hull.end(), hull.end())                             #276
  SegIt it_r1, it_r2;
  function<bool(Seg &, Seg &)> inc_c([&dir](Seg &lft, Seg &rgt) {
    return dot(dir, lft.F) < dot(dir, rgt.F);
  });
  function<bool(Seg &, Seg &)> dec_c([&dir](Seg &lft, Seg &rgt)
  ↪  #431
    returndot(dir, lft.F > dot(dir, rgt.F;
  });
  if (it_min <= it_max) {
    it_r1 = upper_bound(it_min, it_max + 1, l, inc_c) - 1;
    if (dot(dir, hull.front().F) >= opt_val)                      #689
      it_r2 = upper_bound(hull.begin(), it_min + 1, l, dec_c) - 1;
    } else {
      it_r2 = upper_bound(it_max, hull.end(), l, dec_c) - 1;
    }
  } else                                                          #552
    it_r1 = upper_bound(it_max, it_min + 1, l, dec_c) - 1;
    if (dot(dir, hull.front().F) <= opt_val) {
      it_r2 = upper_bound(hull.begin(), it_max + 1, l, inc_c) - 1;
    } else {
      it_r2 = upper_bound(it_min, hull.end(), l, inc_c) - 1       #220
    }
  }
  return MP(it_r1, it_r2);
                                                                  %498
Segdiameter({// O(n)
    Seg res;
    ll dia_sq = 0;
    auto it1 = hull.begin();
    auto it2 = up_beg                                             #632
    Vec v1 = {hull.back().S.F - hull.back().F.F,
      hull.back().S.S - hull.back().F.S};
    while (it2 != hull.begin()) {
      Vec v2 = {(it2 - 1)->S.F - (it2 - 1)->F.F,
        (it2 - 1)->S.S - (it2 - 1)->F.S}                          #150
      if (cross(v1, v2) > 0) break;
      --it2;
    }
    while (it2 != hull.end()) {// check all antipodal pairs
      if (dist_sq(it1->F, it2->F) > dia_sq)                       #246
        res = {it1->F, it2->F};
        dia_sq = dist_sq(res.F, res.S);
      }
      Vec v1 = {it1->S.F - it1->F.F, it1->S.S - it1->F.S};
      Vec v2 = {it2->S.F - it2->F.F, it2->S.S - it2->F.S}         #529
      if (cross(v1, v2) == 0) {
        if (dist_sq(it1->S, it2->F) > dia_sq) {
          res = {it1->S, it2->F};
          dia_sq = dist_sq(res.F, res.S);
                                                                  #406
        if (dist_sq(it1->F, it2->S) > dia_sq) {
          res = {it1->F, it2->S};
          dia_sq = dist_sq(res.F, res.S);
        }// report cross pairs at parallel lines.
        ++it1                                                     #362
        ++it2;
      } else if (cross(v1, v2) < 0) {
        ++it1;
      } else {
        ++it2                                                     #936
      }
    }
    return res;
  }
```

9 Delaunay triangulation $\mathcal{O}(n \log n)$

```
const int max_co = (1 << 28) - 5;
struct Vec {
  int x, y;
  bool operator==(const Vec &oth) { return x == oth.x && y == oth.y; }
  bool operator!=(const Vec &oth) { return !operator==(oth);      #679
  Vec operator-(const Vec &oth) { return {x - oth.x, y - oth.y}; }
};
llcross(Vec a, Vec b{ return (ll)a.x * b.y - (ll)a.y * b.x; }
lldot(Vec a, Vec b{ return (ll)a.x * b.x + (ll)a.y * b.y; }
struct Edge                                                       #732
  Vec tar;
  Edge *nxt;
```

```
  Edge *inv = NULL;
  Edge *rep = NULL;
  bool vis = false                                              #668
};
struct Seg {
  Vec a, b;
  bool operator==(const Seg &oth) { return a == oth.a && b == oth.b; }
  bool operator!=(const Seg &oth) { return !operator==(oth);     #245
};
llorient(Vec a, Vec b, Vec c{
  return (ll)a.x * (b.y - c.y) + (ll)b.x * (c.y - a.y) +
         (ll)c.x * (a.y - b.y);
                                                                %334
boolin_c_circle(Vec *arr, Vec d{
  if (cross(arr[1] - arr[0], arr[2] - arr[0]) == 0)
    return true;// degenerate
  ll m[3][3];
  ran(i, 0, 3)                                                  #264
    m[i][0] = arr[i].x - d.x;
    m[i][1] = arr[i].y - d.y;
    m[i][2] = m[i][0] * m[i][0];
    m[i][2] += m[i][1] * m[i][1];
                                                                #305
  __int128 res = 0;
  res += (__int128)(m[0][0] * m[1][1] - m[0][1] * m[1][0]) * m[2][2];
  res += (__int128)(m[1][0] * m[2][1] - m[1][1] * m[2][0]) * m[0][2];
  res -= (__int128)(m[0][0] * m[2][1] - m[0][1] * m[2][0]) * m[1][2];
  return res > 0                                                #845
                                                                %793
Edge add_triangle(Edge *a, Edge *b, Edge *c{
  Edge *old[] = {a, b, c};
  Edge *tmp = new Edge[3];
  ran(i, 0, 3) {
    old[i]->rep = tmp + i                                       #219
    tmp[i] = {old[i]->tar, tmp + (i + 1) % 3, old[i]->inv};
    if (tmp[i].inv) tmp[i].inv->inv = tmp + i;
  }
  return tmp;
                                                                #178
Edge add_point(Vec p, Edge *cur{// returns outgoing edge
  Edge *triangle[] = {cur, cur->nxt, cur->nxt->nxt};
  ran(i, 0, 3) {
    if (orient(triangle[i]->tar, triangle[(i + 1) % 3]->tar, p) < 0)
      return NULL                                               #233
  }
  ran(i, 0, 3) {
    if (triangle[i]->rep) {
      Edge *res = add_point(p, triangle[i]->rep);
      if (res                                                   #636
        return res;// unless we are on last layer we must exit here
    }
  }
  Edge p_as_e{p};
  Edge tmp{cur->tar}                                            #432
  tmp.inv = add_triangle(&p_as_e, &tmp, cur = cur->nxt);
  Edge *res = tmp.inv->nxt;
  tmp.tar = cur->tar;
  tmp.inv = add_triangle(&p_as_e, &tmp, cur = cur->nxt);
  tmp.tar = cur->tar                                            #359
  res->inv = add_triangle(&p_as_e, &tmp, cur = cur->nxt);
  res->inv->inv = res;
  return res;
}
Edge *delaunay(vector<Vec> &points)                             #029
  random_shuffle(points.begin(), points.end());
  Vec arr[] = {{4 * max_co, 4 * max_co}, {-4 * max_co, max_co},
    {max_co, -4 * max_co}};
  Edge *res = new Edge[3];
  ran(i, 0, 3) res[i] = {arr[i], res + (i + 1) % 3}             #480
  for (Vec &cur : points) {
    Edge *loc = add_point(cur, res);
    Edge *out = loc;
    arr[0] = cur;
    while (true)                                                #601
      arr[1] = out->tar;
      arr[2] = out->nxt->tar;
      Edge *e = out->nxt->inv;
      if (e && in_c_circle(arr, e->nxt->tar)) {
        Edge tmp{cur}                                           #056
        tmp.inv = add_triangle(&tmp, out, e->nxt);
        tmp.tar = e->nxt->tar;
        tmp.inv->inv = add_triangle(&tmp, e->nxt->nxt, out->nxt->nxt);
        out = tmp.inv->nxt;
        continue                                                #173
      }
      out = out->nxt->nxt->inv;
      if (out->tar == loc->tar) break;
    }
                                                                #032
  return res;
                                                                %769
voidextract_triangles(Edge *cur, vector<vector<Seg> > &res{
  if (!cur->vis) {
    bool inc = true;
    Edge *it = cur;
    do                                                          #769
      it->vis = true;
      if (it->rep) {
        extract_triangles(it->rep, res);
        inc = false;
                                                                #104
      it = it->nxt;
```

```
    } while (it != cur);
    if (inc) {
      Edge *triangle[3] = {cur, cur->nxt, cur->nxt->nxt};
      res.resize(res.size() + 1)                                    #207
      vector<Seg> &tar = res.back();
      ran(i, 0, 3) {
        if ((abs(triangle[i]->tar.x) < max_co &&
                abs(triangle[(i + 1) % 3]->tar.x) < max_co))
          tar.push_back                                             #011
            {triangle[i]->tar, triangle[(i + 1) % 3]->tar});
      }
      if (tar.empty()) res.pop_back();
    }
                                                                    #602
```

10 Aho Corasick $\mathcal{O}(|\text{alpha}|\sum \text{len})$

```
const int alpha_size = 26;
struct Node {
  Node *nxt[alpha_size];// May use other structures to move in trie
  Node *suffix;
  Node() { memset(nxt, 0, alpha_size * sizeof(Node *));             #248
  int cnt = 0;
};
Node aho_corasick(vector<vector<char> > &dict{
  Node *root = new Node;
  root->suffix = 0                                                  #292
  vector<pair<vector<char> *, Node *> > state;
  for (vector<char> &s : dict) state.emplace_back(&s, root);
  for (int i = 0; !state.empty(); ++i) {
    vector<pair<vector<char> *, Node *> > nstate;
    for (auto &cur : state)                                         #306
      Node *nxt = cur.second->nxt[(*cur.first)[i]];
      if (nxt) {
        cur.second = nxt;
      } else {
        nxt = new Node                                              #266
        cur.second->nxt[(*cur.first)[i]] = nxt;
        Node *suf = cur.second->suffix;
        cur.second = nxt;
        nxt->suffix = root;// set correct suffix link
        while (suf)                                                 #249
          if (suf->nxt[(*cur.first)[i]]) {
            nxt->suffix = suf->nxt[(*cur.first)[i]];
            break;
          }
          suf = suf->suffix                                         #562
        }
      }
      if (cur.first->size() > i + 1) nstate.push_back(cur);
    }
    state = nstate                                                  #417
  }
  return root;
                    %882 // auxilary functions for searhing and counting
Node walk(Node *cur,
  char c{// longest prefix in dict that is suffix of walked string.
  while (true) {
    if (cur->nxt[c]) return cur->nxt[c];
    if (!cur->suffix) return cur                                    #414
    cur = cur->suffix;
  }
                                                                    %529
voidcnt_matches(Node *root, vector<char> &match_in{
  Node *cur = root;
  for (char c : match_in) {
    cur = walk(cur, c);
    ++cur->cnt                                                      #015
  }
                                                                    %156
voidadd_cnt(Node *root{// After counting matches propagete ONCE to
                        // suffixes for final counts
  vector<Node *> to_visit = {root};
  ran(i, 0, to_visit.size()) {
    Node *cur = to_visit[i];
    ran(j, 0, alpha_size)                                           #662
      if (cur->nxt[j]) to_visit.push_back(cur->nxt[j]);
    }
  }
  for (int i = to_visit.size() - 1; i > 0; --i)
    to_visit[i]->suffix->cnt += to_visit[i]->cnt                    #950
                                                                    %488
intmain({
  int n, len;
  scanf"%d %d", &len, &n);
  vector<char> a(len + 1);
  scanf"%s", a.data());
  a.pop_back();
  for (char &c : a) c -='a';
  vector<vector<char> > dict(n);
  ran(i, 0, n) {
    scanf"%d", &len);
    dict[i].resize(len + 1);
    scanf"%s", dict[i].data());
    dict[i].pop_back();
    for (char &c : dict[i]) c -='a';
  }
  Node *root = aho_corasick(dict);
  cnt_matches(root, a);
  add_cnt(root);
  ran(i, 0, n) {
    Node *cur = root;
    for (char c : dict[i]) cur = walk(cur, c);
    printf"%d\n", cur->cnt);
```

```
}
```

11 Suffix automaton and tree $\mathcal{O}((n+q)\log(|\mathbf{alpha}|))$

```
class Node {
 private:
  map<char, Node *>
    nxt_char;// Map is faster than hashtable and unsorted arrays
 public                                                                  #994
  int len;// Length of longest suffix in equivalence class.
  Node *suf;
  boolhas_nxt(char c const{ return nxt_char.count(c); }
  Node nxt(char c{
    if (!has_nxt(c)) return NULL                                         #788
    return nxt_char[c];
  }
  voidset_nxt(char c, Node *node{ nxt_char[c] = node; }
  Node split(int new_len, char c{
    Node *new_n = new Node                                               #449
    new_n->nxt_char = nxt_char;
    new_n->len = new_len;
    new_n->suf = suf;
    suf = new_n;
    return new_n                                                         #130
                                                                         %044
 // Extra functions for matching and counting
  Node lower_depth(int depth{// move to longest suffix of current
                              // with a maximum length of depth.
    if (suf->len >= depth) return suf->lower_depth(depth);
    return this;
  }
  Node *walk(char c, int depth                                           #736
    int &match_len) {// move to longest suffix of walked path that is
                     // a substring
    match_len = min(match_len,
      len);// includes depth limit(needed for finding matches)
    if (has_nxt(c)) {// as suffixes are in classes match_len must be
                     // tracked externally
      ++match_len                                                        #153
      returnnxt(c->lower_depth(depth;
    }
    if (suf) return suf->walk(c, depth, match_len);
    return this;
                                                                         %969
  int paths_to_end = 0;
  voidset_as_end({// All suffixes of current node are marked as
                  // ending nodes.
    paths_to_end += 1;
    if (suf) suf->set_as_end();
                                                                         #041
  bool vis = false;
  voidcalc_paths_to_end({// Call ONCE from ROOT. For each node
                        // calculates number of ways to reach an
                        // end node.
    if (!vis) {// paths_to_end is ocurence count for any strings in
               // current suffix equivalence class.
      vis = true;
      for (auto cur : nxt_char)                                          #035
        cur.second->calc_paths_to_end();
        paths_to_end += cur.second->paths_to_end;
      }
    }
                                                                         %996
 // Transform into suffix tree of reverse string
  map<char, Node *> tree_links;
  int end_dist = 1 << 30;
  intcalc_end_dist({
    if (end_dist == 1 << 30) {
      if (nxt_char.empty()) end_dist = 0                                 #524
      for (auto cur : nxt_char)
        end_dist = min(end_dist, 1 + cur.second->calc_end_dist());
    }
    return end_dist;
                                                                         #021
  bool vis_t = false;
  voidbuild_suffix_tree(string &s{// Call ONCE from ROOT.
    if (!vis_t) {
      vis_t = true;
      if (suf                                                            #270
        suf->tree_links[s[s.size() - end_dist - suf->len - 1]] = this;
      for (auto cur : nxt_char) cur.second->build_suffix_tree(s);
    }
  }
}                                                                        #268
struct SufAuto {
  Node *last;
  Node *root;
  voidextend(char new_c{
    Node *new_end = new Node                                             #340
    new_end->len = last->len + 1;
    Node *suf_w_nxt = last;
    while (suf_w_nxt && !suf_w_nxt->has_nxt(new_c)) {
      suf_w_nxt->set_nxt(new_c, new_end);
      suf_w_nxt = suf_w_nxt->suf                                         #217
    }
    if (!suf_w_nxt) {
      new_end->suf = root;
    } else {
      Node *max_sbstr = suf_w_nxt->nxt(new_c)                            #618
      if (suf_w_nxt->len + 1 == max_sbstr->len) {
        new_end->suf = max_sbstr;
      } else {
        Node *eq_sbstr = max_sbstr->split(suf_w_nxt->len + 1, new_c);
        new_end->suf = eq_sbstr                                          #295
```

```
      Node *w_edge_to_eq_sbstr = suf_w_nxt;
      while (w_edge_to_eq_sbstr != 0 &&
             w_edge_to_eq_sbstr->nxt(new_c) == max_sbstr) {
        w_edge_to_eq_sbstr->set_nxt(new_c, eq_sbstr);
        w_edge_to_eq_sbstr = w_edge_to_eq_sbstr->suf          #678
      }
    }
  }
  last = new_end;
                                                              %135
SufAuto(string &s) {
  root = new Node;
  root->len = 0;
  root->suf = NULL;
  last = root                                                 #604
  for (char c : s) extend(c);
  root->calc_end_dist();// To build suffix tree use reversed string
  root->build_suffix_tree(s);
}
```

12 Dinic

```
struct MaxFlow {
  typedef long long ll;
  const ll INF = 1e18;
  struct Edge {
    int u, v                                                  #295
    ll c, rc;
    shared_ptr<ll> flow;
    Edge(int _u, int _v, ll _c, ll _rc = 0)
      : u(_u), v(_v), c(_c), rc(_rc) {}
  }                                                           #787
  struct FlowTracker {
    shared_ptr<ll> flow;
    ll cap, rcap;
    bool dir;
    FlowTracker(ll _cap, ll _rcap, shared_ptr<ll> _flow, int _dir  #418
    ↪
      : cap(_cap), rcap(_rcap), flow(_flow), dir(_dir) {}
    ll rem() const {
      if (dir == 0) {
        return cap - *flow;
      } else                                                  #844
        return rcap + *flow;
      }
    }
    void add_flow(ll f) {
      if (dir == 0                                            #920
        *flow += f;
      else
        *flow -= f;
      assert(*flow <= cap);
      assert(-*flow <= rcap)                                  #287
    }
    operator ll() const { return rem(); }
    void operator-=(ll x) { add_flow(x); }
    void operator+=(ll x) { add_flow(-x); }
  }                                                           #789
  int source, sink;
  vector<vector<int> > adj;
  vector<vector<FlowTracker> > cap;
  vector<Edge> edges;
  MaxFlow(int _source, int _sink) : source(_source), sink(_sink)   #080
  ↪
    assert(source != sink);
  }
  int add_edge(int u, int v, ll c, ll rc = 0) {
    edges.push_back(Edge(u, v, c, rc));
    return edges.size() - 1                                   #659
  }
  vector<int> now, lvl;
  voidprep({
    int max_id = max(source, sink);
    for (auto edge : edges) max_id = max(max_id, max(edge.u, edge.v))
    ↪  #638
    adj.resize(max_id + 1);
    cap.resize(max_id + 1);
    now.resize(max_id + 1);
    lvl.resize(max_id + 1);
    for (auto &edge : edges)                                  #604
      auto flow = make_shared<ll>(0);
      adj[edge.u].push_back(edge.v);
      cap[edge.u].push_back(FlowTracker(edge.c, edge.rc, flow, 0));
      if (edge.u != edge.v) {
        adj[edge.v].push_back(edge.u)                         #789
        cap[edge.v].push_back(FlowTracker(edge.c, edge.rc, flow, 1));
      }
      assert(cap[edge.u].back() == edge.c);
      edge.flow = flow;
                                                              #131
  }
  booldinic_bfs({
    fill(now.begin(), now.end(), 0);
    fill(lvl.begin(), lvl.end(), 0);
    lvl[source] = 1                                           #448
    vector<int> bfs(1, source);
    for (int i = 0; i < bfs.size(); ++i) {
      int u = bfs[i];
      for (int j = 0; j < adj[u].size(); ++j) {
        int v = adj[u][j]                                     #227
        if (cap[u][j] > 0 && lvl[v] == 0) {
          lvl[v] = lvl[u] + 1;
          bfs.push_back(v);
        }
```

```
                                                                    #722
    }
    return lvl[sink] > 0;
  }
  lldinic_dfs(int u, ll flow{
    if (u == sink) return flow                                      #725
    while (now[u] < adj[u].size()) {
      int v = adj[u][now[u]];
      if (lvl[v] == lvl[u] + 1 && cap[u][now[u]] != 0) {
        ll res = dinic_dfs(v, min(flow, (ll)cap[u][now[u]]));
        if (res > 0)                                                #459
          cap[u][now[u]] -= res;
          return res;
        }
      }
      ++now[u]                                                      #460
    }
    return 0;
  }
  llcalc_max_flow({
    prep()                                                          #054
    ll ans = 0;
    while (dinic_bfs()) {
      ll cur = 0;
      do {
        cur = dinic_dfs(source, INF)                                #052
        ans += cur;
      } while (cur > 0);
    }
    return ans;
                                                                    #346
  llflow_on_edge(int edge_index{
    assert(edge_index < edges.size());
    return *edges[edge_index].flow;
  }
}                                                                   %583
intmain({
  int n, m;
  cin >> n >> m;
  auto mf = MaxFlow(
    1, n);// arguments source and sink, memory usage O(largest node
          // index + input size), sink doesn't need to be last index
  int edge_index;
  for (int i = 0; i < m; ++i) {
    int a, b, c;
    cin >> a >> b >> c;
   // mf.add_edge(a,b,c); // for directed edges
    edge_index = mf.add_edge(
      a, b, c, c);// store edge index if care about flow value
  }
  cout << mf.calc_max_flow() <<'\n';
 // cout << mf.flow_on_edge(edge_index) << endl; // return flow on
 // this edge
```

13 Min Cost Max Flow with succesive dijkstra $\mathcal{O}(\text{flow} \cdot n^2)$

```
const int nmax = 1055;
const ll inf = 1e14;
int t, n, v;// 0 is source, v-1 sink
ll rem_flow[nmax][nmax];// set [x][y] for directed capacity from x to y.
ll cost[nmax][nmax];// set [x][y] for directed cost from x to y. SET   #497
                    // TO inf IF NOT USED
ll min_dist[nmax];
int prev_node[nmax];
ll node_flow[nmax];
bool visited[nmax]                                                    %576
ll tot_cost, tot_flow;// output
voidmin_cost_max_flow({
  tot_cost = 0;// Does not work with negative cycles.
  tot_flow = 0;
  ll sink_pot = 0                                                     #614
  min_dist[0] = 0                                                     %927
  for (int i = 1; i <= v; ++i) {// incase of no negative edges
                                // Bellman-Ford can be removed.
    min_dist[i] = inf;
  }
  for (int i = 0; i < v - 1; ++i) {
    for (int j = 0; j < v; ++j)                                       #654
      for (int k = 0; k < v; ++k) {
        if (rem_flow[j][k] > 0 &&
            min_dist[j] + cost[j][k] < min_dist[k])
          min_dist[k] = min_dist[j] + cost[j][k];
                                                                      #040
    }
  }
  for (int i = 0; i < v; ++i) {// Apply potentials to edge costs.
    for (int j = 0; j < v; ++j) {
      if (cost[i][j] != inf)                                          #037
        cost[i][j] += min_dist[i];
        cost[i][j] -= min_dist[j];
      }
    }
                                                                      #630
  sink_pot += min_dist[v - 1];// Bellman-Ford end.                    %849
  while (true) {
    for (int i = 0; i <= v; ++i) {// node after sink is used as start
                                  // value for Dijkstra.
      min_dist[i] = inf;
      visited[i] = false;
                                                                      #827
    min_dist[0] = 0;
    node_flow[0] = inf;
```

```
    int min_node;
    while (true) {// Use Dijkstra to calculate potentials
      int min_node = v                                          #782
      for (int i = 0; i < v; ++i) {
        if ((!visited[i]) && min_dist[i] < min_dist[min_node])
          min_node = i;
      }
      if (min_node == v) break visited[min_node] = true         #281
      for (int i = 0; i < v; ++i) {
        if ((!visited[i]) &&
            min_dist[min_node] + cost[min_node][i] < min_dist[i]) {
          min_dist[i] = min_dist[min_node] + cost[min_node][i];
          prev_node[i] = min_node                               #881
          node_flow[i] =
            min(node_flow[min_node], rem_flow[min_node][i]);
        }
      }
                                                                #046
    if (min_dist[v - 1] == inf)
      break for (int i = 0; i < v;
                ++i) {// Apply potentials to edge costs.
        for (int j = 0; j < v;
             ++j) {// Found path from source to sink becomes 0 cost.   #664
          if (cost[i][j] != inf) {
            cost[i][j] += min_dist[i];
            cost[i][j] -= min_dist[j];
          }
                                                                #983
      }
    sink_pot += min_dist[v - 1];
    tot_flow += node_flow[v - 1];
    tot_cost += sink_pot * node_flow[v - 1];
    int cur = v - 1                                             #946
    while (cur != 0) {
     // Backtrack along found path that now has 0 cost.
      rem_flow[prev_node[cur]][cur] -= node_flow[v - 1];
      rem_flow[cur][prev_node[cur]] += node_flow[v - 1];
      cost[cur][prev_node[cur]] = 0;
      if (rem_flow[prev_node[cur]][cur] == 0                    #446
        cost[prev_node[cur]][cur] = inf;
      cur = prev_node[cur];
    }
  }
                                                                %803
intmain({// http://www.spoj.com/problems/GREED/
  cin >> t;
  for (int i = 0; i < t; ++i) {
    cin >> n;
    for (int j = 0; j < nmax; ++j) {
      for (int k = 0; k < nmax; ++k) {
        cost[j][k] = inf;
        rem_flow[j][k] = 0;
      }
    }
    for (int j = 1; j <= n; ++j) {
      cost[j][2 * n + 1] = 0;
      rem_flow[j][2 * n + 1] = 1;
    }
    for (int j = 1; j <= n; ++j) {
      int card;
      cin >> card;
      ++rem_flow[0][card];
      cost[0][card] = 0;
    }
    int ex_c;
    cin >> ex_c;
    for (int j = 0; j < ex_c; ++j) {
      int a, b;
      cin >> a >> b;
      if (b < a) swap(a, b);
      cost[a][b] = 1;
      rem_flow[a][b] = nmax;
      cost[b][n + b] = 0;
      rem_flow[b][n + b] = nmax;
      cost[n + b][a] = 1;
      rem_flow[n + b][a] = nmax;
    }
    v = 2 * n + 2;
    min_cost_max_flow();
    cout << tot_cost <<'\n';
  }
```

14 Min Cost Max Flow with Cycle Cancelling $\mathcal{O}(\mathbf{flow} \cdot nm)$

```
struct Network {
  struct Node;
  struct Edge {
    Node *u, *v;
    int f, c, cost                                              #965
    Node*from(Node* pos{
      if (pos == u) return v;
      return u;
    }
    intgetCap(Node* pos                                         #145
      if (pos == u) return c - f;
      return f;
    }
    int addFlow(Node* pos, int toAdd) {
      if (pos == u)                                             #369
        f += toAdd;
        return toAdd * cost;
      } else {
        f -= toAdd;
```

```
      return -toAdd * cost                    #987
    }
  }
};
struct Node {
  vector<Edge*> conn                          #577
  int index;
};
deque<Node> nodes;
deque<Edge> edges;
Node*addNode(                                 #057
  nodes.push_back(Node());
  nodes.back().index = nodes.size() - 1;
  return &nodes.back();
}
Edge*addEdge(Node* u, Node* v, int f, int c, int cost    #518
  edges.push_back({u, v, f, c, cost});
  u->conn.push_back(&edges.back());
  v->conn.push_back(&edges.back());
  return &edges.back();
                                              #692
// Assumes all needed flow has already been added
intminCostMaxFlow({
  int n = nodes.size();
  int result = 0;
  struct State {
    int p                                     #091
    Edge* used;
  };
  while (1) {
    vector<vector<State> > state(1, vector<State>(n, {0, 0}));
    for (int lev = 0; lev < n; lev++)         #158
      state.push_back(state[lev]);
      for (int i = 0; i < n; i++) {
        if (lev == 0 || state[lev][i].p < state[lev - 1][i].p) {
          for (Edge* edge : nodes[i].conn) {
            if (edge->getCap(&nodes[i]) > 0)  #760
              int np =
                state[lev][i].p +
                (edge->u == &nodes[i] ? edge->cost : -edge->cost);
              int ni = edge->from(&nodes[i])->index;
              if (np < state[lev + 1][ni].p)  #281
                state[lev + 1][ni].p = np;
                state[lev + 1][ni].used = edge;
              }
            }
                                              #460
        }
      }
    }
    // Now look at the last level
    bool valid = false;
    for (int i = 0; i < n; i++                #283
      if (state[n - 1][i].p > state[n][i].p) {
        valid = true;
        vector<Edge*> path;
        int cap = 1000000000;
        Node* cur = &nodes[i]                 #352
        int clev = n;
        vector<bool> explr(n, false);
        while (!explr[cur->index]) {
          explr[cur->index] = true;
          State cstate = state[clev][cur->index]    #954
          cur = cstate.used->from(cur);
          path.push_back(cstate.used);
        }
        reverse(path.begin(), path.end());
                                              #592
          int i = 0;
          Node* cur2 = cur;
          do {
            cur2 = path[i]->from(cur2);
            i++                               #990
          } while (cur2 != cur);
          path.resize(i);
        }
        for (auto edge : path) {
          cap = min(cap, edge->getCap(cur))   #297
          cur = edge->from(cur);
        }
        for (auto edge : path) {
          result += edge->addFlow(cur, cap);
          cur = edge->from(cur)               #599
        }
      }
    if (!valid) break;
  }
  return result                               #550
}
```

15 DMST $\mathcal{O}(E \log V)$

```
struct EdgeDesc {
  int from, to, w;
};
struct DMST {
  struct Node                                 #091
  struct Edge {
    Node *from;
    Node *tar;
    int w;
    bool inc                                  #186
  };
  struct Circle {
```

```
  bool vis = false;
  vector<Edge *> contents;
  voidclean(int idx                                          #946
};
const static greater<pair<ll, Edge *> >
  comp;// Can use inline static since C++17
static vector<Circle> to_process;
static bool no_dmst                                          #478
static Node *root;
struct Node {
  Node *par = NULL;
  vector<pair<int, int> > out_cands;// Circ, edge idx
  vector<pair<ll, Edge *> > con                              #608
  bool in_use = false;
  ll w = 0;// extra to add to edges in con
  Nodeanc({
    if (!par return thi;
    while (par->par) par = par->par                          #721
    return par;
  }
  voidclean({
    if (!no_dmst) {
      in_use = false                                         #465
      for (auto &cur : out_cands)
        to_process[cur.first].clean(cur.second);
    }
  }
  Node con_to_root(                                          #488
    if (anc() == root) return root;
    in_use = true;
    Node *super = this;// Will become root or the first Node
                       // encountered in a loop.
    while (super == this) {
      while                                                  #363
        !con.empty() && con.front().second->tar->anc() == anc()) {
        pop_heap(con.begin(), con.end(), comp);
        con.pop_back();
      }
      if (con.empty())                                       #506
        no_dmst = true;
        return root;
      }
      pop_heap(con.begin(), con.end(), comp);
      auto nxt = con.back()                                  #541
      con.pop_back();
      w = -nxt.first;
      if (nxt.second->tar
            ->in_use) {// anc() wouldn't change anything
        super = nxt.second->tar->anc()                       #174
        to_process.resize(to_process.size() + 1);
      } else {
        super = nxt.second->tar->con_to_root();
      }
      if (super != root)                                     #595
        to_process.back().contents.push_back(nxt.second);
        out_cands.emplace_back(to_process.size() - 1,
          to_process.back().contents.size() - 1);
      } else {// Clean circles
        nxt.second->inc = true                               #848
        nxt.second->from->clean();
      }
    }
    if (super != root) {// we are some loops non first Node.
      if (con.size() > super->con.size())                    #860
        swap(con,
          super->con);// Largest con in loop should not be copied.
        swap(w, super->w);
      }
      for (auto cur : con)                                   #064
        super->con.emplace_back(
          cur.first - super->w + w, cur.second);
        push_heap(super->con.begin(), super->con.end(), comp);
      }
                                                             #295
    par = super;// root or anc() of first Node encountered in a
                // loop
    return super;
  }
};
Node *cur_root                                               #995
vector<Node> graph;
vector<Edge> edges;
DMST(int n, vector<EdgeDesc> &desc,
  int r) {// Self loops and multiple edges are okay.
  graph.resize(n)                                            #989
  cur_root = &graph[r];
  for (auto &cur : desc)// Edges are reversed internally
    edges.push_back(Edge{&graph[cur.to], &graph[cur.from], cur.w});
  for (int i = 0; i < desc.size(); ++i)
    graph[desc[i].to].con.emplace_back(desc[i].w, &edges[i])    #895
  for (int i = 0; i < n; ++i)
    make_heap(graph[i].con.begin(), graph[i].con.end(), comp);
}
bool find() {
  root = cur_root                                            #771
  no_dmst = false;
  for (auto &cur : graph) {
    cur.con_to_root();
    to_process.clear();
    if (no_dmst) return false                                #405
  }
  return true;
                                                             %732
```

```
  llweight({
    ll res = 0;
    for (auto &cur : edges) {
      if (cur.inc) res += cur.w;
                                                              #369
    return res;
                                                              %477
};
void DMST::Circle::clean(int idx) {
  if (!vis) {
    vis = true;
    for (int i = 0; i < contents.size(); ++i)                 #814
      if (i != idx) {
        contents[i]->inc = true;
        contents[i]->from->clean();
      }
                                                              #711
  }
}
const greater<pair<ll, DMST::Edge *> > DMST::comp;
vector<DMST::Circle> DMST::to_process;
bool DMST::no_dmst                                            #417
```

16 Bridges $\mathcal{O}(n)$

```
struct vert;
struct edge {
  bool exists = true;
  vert *dest;
  edge *rev                                                   #922
  edge(vert *_dest) : dest(_dest) { rev = NULL; }
  vert &operator*() { return *dest; }
  vert *operator->() { return dest; }
  bool is_bridge();
}                                                             #116
struct vert {
  deque<edge> con;
  int val = 0;
  int seen;
  intdfs(int upd, edge *ban{// handles multiple edges         #331
    if (!val) {
      val = upd;
      seen = val;
      for (edge &nxt : con) {
        if (nxt.exists && (&nxt) != ban                       #866
          seen = min(seen, nxt->dfs(upd + 1, nxt.rev));
      }
    }
    return seen;
                                                              %624
  voidremove_adj_bridges({
    for (edge &nxt : con) {
      if (nxt.is_bridge()) nxt.exists = false;
    }
                                                              %106
  intcnt_adj_bridges({
    int res = 0;
    for (edge &nxt : con) res += nxt.is_bridge();
    return res;
                                                              %056
};
bool edge::is_bridge() {
  return exists &&
        (dest->seen > rev->dest->val || dest->val < rev->dest->seen);
                                                              %223
vert graph[nmax];
intmain({// Mechanics Practice BRIDGES
  int n, m;
  cin >> n >> m;
  for (int i = 0; i < m; ++i) {
    int u, v;
    scanf"%d %d", &u, &v);
    graph[u].con.emplace_back(graph + v);
    graph[v].con.emplace_back(graph + u);
    graph[u].con.back().rev = &graph[v].con.back();
    graph[v].con.back().rev = &graph[u].con.back();
  }
  graph[1].dfs(1, NULL);
  int res = 0;
  for (int i = 1; i <= n; ++i) res += graph[i].cnt_adj_bridges();
  cout << res / 2 << endl;
```

17 2-Sat $\mathcal{O}(n)$ and SCC $\mathcal{O}(n)$

```
struct Graph {
  int n;
  vector<vector<int> > conn;
  Graph(int nsize) {
    n = nsize                                                 #987
    conn.resize(n);
  }
  void add_edge(int u, int v) { conn[u].push_back(v); }
  void _topsort_dfs(int pos, vector<int> &result, vector<bool> &explr,
    vector<vector<int> > &revconn)                            #592
    if (explr[pos]) return;
    explr[pos] = true;
    for (auto next : revconn[pos])
      _topsort_dfs(next, result, explr, revconn);
    result.push_back(pos)                                     #810
  }
  vector<int> topsort() {
    vector<vector<int> > revconn(n);
    for (int u = 0; u < n; u++) {
      for (auto v : conn[u]) revconn[v].push_back(u)          #775
    }
    vector<int> result;
```

```
    vector<bool> explr(n, false);
    for (int i = 0; i < n; i++)
      _topsort_dfs(i, result, explr, revconn)                          #178
    reverse(result.begin(), result.end());
    return result;
  }
  void dfs(int pos, vector<int> &result, vector<bool> &explr) {
    if (explr[pos]) return                                             #591
    explr[pos] = true;
    for (auto next : conn[pos]) dfs(next, result, explr);
    result.push_back(pos);
                                                                       %603
  vector<vector<int> > scc() {
    vector<int> order = topsort();
    reverse(order.begin(), order.end());
    vector<bool> explr(n, false);
    vector<vector<int> > results                                       #020
    for (auto it = order.rbegin(); it != order.rend(); ++it) {
      vector<int> component;
      _topsort_dfs(*it, component, explr, conn);
      sort(component.begin(), component.end());
      results.push_back(component)                                     #741
    }
    sort(results.begin(), results.end());
    return results;
  }
}                                                   %983 // Solution for:
// http://codeforces.com/group/PjzGiggT71/contest/221700/problem/C
intmain({
  int n, m;
  cin >> n >> m;
  Graphg(2 * m;
  for (int i = 0; i < n; i++) {
    int a, sa, b, sb;
    cin >> a >> sa >> b >> sb;
    a--, b--;
    g.add_edge(2 * a + 1 - sa, 2 * b + sb);
    g.add_edge(2 * b + 1 - sb, 2 * a + sa);
  }
  vector<int> state(2 * m, 0);
  {
    vector<int> order = g.topsort();
    vector<bool> explr(2 * m, false);
    for (auto u : order) {
      vector<int> traversed;
      g.dfs(u, traversed, explr);
      if (traversed.size() > 0 && !state[traversed[0] ^ 1]) {
        for (auto c : traversed) state[c] = 1;
      }
    }
  }
  for (int i = 0; i < m; i++) {
    if (state[2 * i] == state[2 * i + 1]) {
      cout <<"IMPOSSIBLE\n";
      return 0;
    }
  }
  for (int i = 0; i < m; i++) {
    cout << state[2 * i + 1] <<'\n';
  }
  return 0;
```

18 Generic persistent compressed lazy segment tree

```
struct Seg {
  ll sum = 0;
  voidrecalc(const Seg &lhs_seg, int lhs_len, const Seg &rhs_seg,
    int rhs_len{
    sum = lhs_seg.sum + rhs_seg.sum                                    #684
  }
} __attribute__((packed));
struct Lazy {
  ll add;
  ll assign_val;// LLONG_MIN if no assign;                             #529
  voidinit({
    add = 0;
    assign_val = LLONG_MIN;
  }
  Lazy() { init();                                                     #819
  voidsplit(Lazy &lhs_lazy, Lazy &rhs_lazy, int len{
    lhs_lazy = *this;
    rhs_lazy = *this;
    init();
                                                                       #953
  voidmerge(Lazy &oth, int len{
    if (oth.assign_val != LLONG_MIN) {
      add = 0;
      assign_val = oth.assign_val;
                                                                       #949
    add += oth.add;
  }
  void apply_to_seg(Seg &cur, int len) const {
    if (assign_val != LLONG_MIN) {
      cur.sum = len * assign_val                                       #204
    }
    cur.sum += len * add;
  }
} __attribute__((packed))                                              %625
struct Node {// Following code should not need to be modified
  int ver;
  bool is_lazy = false;
  Seg seg;
  Lazy lazy                                                            #321
  Node *lc = NULL, *rc = NULL;
```

```
voidinit({
  if (!lc) {
    lc = new Node{ver};
    rc = new Node{ver}                                          #313
  }
}
Node upd(int L, int R, int l, int r, Lazy &val, int tar_ver{
  if (ver != tar_ver) {
    Node *rep = new Node(*this)                                 #874
    rep->ver = tar_ver;
    return rep->upd(L, R, l, r, val, tar_ver);
  }
  if (L >= l && R <= r) {
    val.apply_to_seg(seg, R - L)                                #138
    lazy.merge(val, R - L);
    is_lazy = true;
  } else {
    init();
    int M = (L + R) / 2                                         #209
    if (is_lazy) {
      Lazy l_val, r_val;
      lazy.split(l_val, r_val, R - L);
      lc = lc->upd(L, M, L, M, l_val, ver);
      rc = rc->upd(M, R, M, R, r_val, ver)                      #104
      is_lazy = false;
    }
    Lazy l_val, r_val;
    val.split(l_val, r_val, R - L);
    if (l < M) lc = lc->upd(L, M, l, r, l_val, ver)             #045
    if (M < r) rc = rc->upd(M, R, l, r, r_val, ver);
    seg.recalc(lc->seg, M - L, rc->seg, R - M);
  }
  return this;
                                                                #441
voidget(int L, int R, int l, int r, Seg *&lft_res, Seg *&tmp,
  bool last_ver{
  if (L >= l && R <= r) {
    tmp->recalc(*lft_res, L - l, seg, R - L);
    swap(lft_res, tmp)                                          #394
  } else {
    init();
    int M = (L + R) / 2;
    if (is_lazy) {
      Lazy l_val, r_val                                         #803
      lazy.split(l_val, r_val, R - L);
      lc = lc->upd(L, M, L, M, l_val, ver + last_ver);
      lc->ver = ver;
      rc = rc->upd(M, R, M, R, r_val, ver + last_ver);
      rc->ver = ver                                             #593
      is_lazy = false;
    }
    if (l < M) lc->get(L, M, l, r, lft_res, tmp, last_ver);
    if (M < r) rc->get(M, R, l, r, lft_res, tmp, last_ver);
                                                                #770
  }
} __attribute__((packed));
struct SegTree {        // indexes start from 0, ranges are [beg, end)
  vector<Node *> roots;// versions start from 0
  int len                                                       #873
  SegTree(int _len) : len(_len) { roots.push_back(new Node{0}); }
  int upd(int l, int r, Lazy &val, bool new_ver = false) {
    Node *cur_root =
      roots.back()->upd(0, len, l, r, val, roots.size() - !new_ver);
    if (cur_root != roots.back()) roots.push_back(cur_root)     #700
    return roots.size() - 1;
  }
  Seg get(int l, int r, int ver = -1) {
    if (ver == -1) ver = roots.size() - 1;
    Seg seg1, seg2                                              #751
    Seg *pres = &seg1, *ptmp = &seg2;
    roots[ver]->get(0, len, l, r, pres, ptmp, roots.size() - 1);
    return *pres;
  }
}                                     #542                      %542
intmain({
  int n, m;// solves Mechanics Practice LAZY
  cin >> n >> m;
  SegTreeseg_tree(1 << 17;
  for (int i = 0; i < n; ++i) {
    Lazy tmp;
    scanf"%lld", &tmp.assign_val);
    seg_tree.upd(i, i + 1, tmp);
  }
  for (int i = 0; i < m; ++i) {
    int o;
    int l, r;
    scanf"%d %d %d", &o, &l, &r);
    --l;
    if (o == 1) {
      Lazy tmp;
      scanf"%lld", &tmp.add);
      seg_tree.upd(l, r, tmp);
    } else if (o == 2) {
      Lazy tmp;
      scanf"%lld", &tmp.assign_val);
      seg_tree.upd(l, r, tmp);
    } else {
      Seg res = seg_tree.get(l, r);
      printf"%lld\n", res.sum);
    }
  }
```

19 Templated HLD $\mathcal{O}(M(n) \log n)$ per query

```
class dummy {
 public:
  dummy() {}
  dummy(int, int) {}
  void set(int, int) {                                          #531
  intquery(int left, int right{
    cout << this <<' ' << left <<' ' << right << endl;
  }
}      %932 /* T should be the type of the data stored in each vertex;
 * DS should be the underlying data structure that is used to peform
 * the group operation. It should have the following methods:
 * * DS () - empty constructor
 * * DS (int size, T initial) - constructs the structure with the
 * given size, initially filled with initial.
 * * void set (int index, T value) - set the value at index `index` to
 * `value`
 * * T query (int left, int right) - return the "sum" of elements
 * between left and right, inclusive.
 */
template <typename T, class DS>
class HLD {
  int vertexc;
  vector<int> *adj;
  vector<int> subtree_size                                      #178
  DS structure;
  DS aux;
  voidbuild_sizes(int vertex, int parent{
    subtree_size[vertex] = 1;
    for (int child : adj[vertex])                               #037
      if (child != parent) {
        build_sizes(child, vertex);
        subtree_size[vertex] += subtree_size[child];
      }
                                                                #759
  }
  int cur;
  vector<int> ord;
  vector<int> chain_root;
  vector<int> par                                               #593
  voidbuild_hld(int vertex, int parent, int chain_source{
    cur++;
    ord[vertex] = cur;
    chain_root[vertex] = chain_source;
    par[vertex] = parent                                        #432
    if (adj[vertex].size() > 1 ||
        (vertex == 1 && adj[vertex].size() == 1)) {
      int big_child, big_size = -1;
      for (int child : adj[vertex]) {
        if ((child != parent) && (subtree_size[child] > big_size))
        ↪  #042
          big_child = child;
          big_size = subtree_size[child];
        }
      }
      build_hld(big_child, vertex, chain_source)                #254
      for (int child : adj[vertex]) {
        if ((child != parent) && (child != big_child))
          build_hld(child, vertex, child);
      }
                                                                #461
  }
public:
  HLD(int _vertexc) {
    vertexc = _vertexc;
    adj = new vector<int>[vertexc + 5]                          #800
  }
  void add_edge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
                                                                #587
  voidbuild(T initial{
    subtree_size = vector<int>(vertexc + 5);
    ord = vector<int>(vertexc + 5);
    chain_root = vector<int>(vertexc + 5);
    par = vector<int>(vertexc + 5)                              #976
    cur = 0;
    build_sizes(1, -1);
    build_hld(1, -1, 1);
    structure = DS(vertexc + 5, initial);
    aux = DS(50, initial)                                       #638
  }
  void set(int vertex, int value) {
    structure.set(ord[vertex], value);
  }
  T query_path                                                  #325
    int u, int v) {/* returns the "sum" of the path u->v */
    int cur_id = 0;
    while (chain_root[u] != chain_root[v]) {
      if (ord[u] > ord[v]) {
        cur_id++                                                #052
        aux.set(cur_id, structure.query(ord[chain_root[u]], ord[u]));
        u = par[chain_root[u]];
      } else {
        cur_id++;
        aux.set(cur_id, structure.query(ord[chain_root[v]], ord[v]))
        ↪  #485
        v = par[chain_root[v]];
      }
    }
    cur_id++;
    aux.set(cur_id                                              #041
```

```
    structure.query(min(ord[u], ord[v]), max(ord[u], ord[v])));
  return aux.query(1, cur_id);
                                                            %905
 voidprint({
  for (int i = 1; i <= vertexc; i++)
    cout << i <<' ' << ord[i] <<' ' << chain_root[i] <<' '
         << par[i] << endl;
 }
};
intmain({
 int vertexc;
 cin >> vertexc;
 HLD<int, dummy> hld(vertexc);
 for (int i = 0; i < vertexc - 1; i++) {
  int u, v;
  cin >> u >> v;
  hld.add_edge(u, v);
 }
 hld.build(0);
 hld.print();
 int queryc;
 cin >> queryc;
 for (int i = 0; i < queryc; i++) {
  int u, v;
  cin >> u >> v;
  hld.query_path(u, v);
  cout << endl;
 }
```

20 Templated multi dimensional BIT $\mathcal{O}(\log(n)^{\textbf{dim}})$ per query

```
// Fully overloaded any dimensional BIT, use any type for coordinates,
// elements, return_value. Includes coordinate compression.
template <typename elem_t, typename coord_t, coord_t n_inf,
  typename ret_t>
class BIT {
 vector<coord_t> positions;
 vector<elem_t> elems                                       #324
 bool initiated = false;
 public:
 BIT() { positions.push_back(n_inf); }
 void initiate() {
  if (initiated)                                            #330
   for (elem_t &c_elem : elems) c_elem.initiate();
  } else {
   initiated = true;
   sort(positions.begin(), positions.end());
   positions.resize(unique(positions.begin(), positions.end())
   ↪ #822
                    positions.begin());
   elems.resize(positions.size());
  }
 }
 template <typename... loc_form                             #620
 voidupdate(coord_t cord, loc_form... args{
  if (initiated) {
   int pos =
    lower_bound(positions.begin(), positions.end(), cord) -
    positions.begin()                                       #346
   for (; pos < positions.size(); pos += pos & -pos)
    elems[pos].update(args...);
  } else {
   positions.push_back(cord);
                                                            #542
 }
 template <typename... loc_form>
 ret_t query(coord_t cord,
  loc_form... args) {// sum in open interval (-inf, cord)
  ret_t res = 0                                             #326
  int pos = (lower_bound(positions.begin(), positions.end(), cord) -
              positions.begin()) -
             1;
  for (; pos > 0; pos -= pos & -pos)
   res += elems[pos].query(args...)                         #549
  return res;
 }
};
template <typename internal_type>
struct wrapped                                              #616
 internal_type a = 0;
 voidupdate(internal_type b{ a += b; }
 internal_typequery({ return a; }
 // Should never be called, needed for compilation
 voidinitiate({ cerr <<'i' << endl; }
 voidupdate({ cerr <<'u' << endl;                           #636
}                                                           %714
intmain({
 // retun type should be same as type inside wrapped
 BIT<BIT<wrapped<ll>, int, INT_MIN, ll>, int, INT_MIN, ll> fenwick;
 int dim = 2;
 vector<tuple<int, int, ll> > to_insert;
 to_insert.emplace_back(1, 1, 1);
 // set up all positions that are to be used for update
 for (int i = 0; i < dim; ++i) {
  for (auto &cur : to_insert)
   fenwick.update(get<0>(cur),
    get<1>(cur));// May include value which won't be used
  fenwick.initiate();
 }
 // actual use
 for (auto &cur : to_insert)
  fenwick.update(get<0>(cur), get<1>(cur), get<2>(cur));
 cout << fenwick.query(2, 2) <<'\n';
```

21 Treap $\mathcal{O}(\log n)$ per query

```cpp
mt19937 randgen;
struct Treap {
  struct Node {
    int key;
    int value                                               #615
    unsigned int priority;
    long long total;
    Node* lch;
    Node* rch;
    Node(int new_key, int new_value)                        #698
      key = new_key;
      value = new_value;
      priority = randgen();
      total = new_value;
      lch = 0                                               #232
      rch = 0;
    }
    void update() {
      total = value;
      if (lch) total += lch->total                          #295
      if (rch) total += rch->total;
    }
  };
  deque<Node> nodes;
  Node* root = 0                                            #633
  pair<Node*, Node*> split(int key, Node* cur) {
    if (cur == 0) return {0, 0};
    pair<Node*, Node*> result;
    if (key <= cur->key) {
      auto ret = split(key, cur->lch)                       #233
      cur->lch = ret.second;
      result = {ret.first, cur};
    } else {
      auto ret = split(key, cur->rch);
      cur->rch = ret.first                                  #988
      result = {cur, ret.second};
    }
    cur->update();
    return result;
                                                            #230
  Node*merge(Node* left, Node* right{
    if (left == 0) return right;
    if (right == 0) return left;
    Node* top;
    if (left->priority < right->priority)                   #282
      left->rch = merge(left->rch, right);
      top = left;
    } else {
      right->lch = merge(left, right->lch);
      top = right                                           #510
    }
    top->update();
    return top;
  }
  void insert(int key, int value)                           #918
    nodes.push_back(Node(key, value));
    Node* cur = &nodes.back();
    pair<Node*, Node*> ret = split(key, root);
    cur = merge(ret.first, cur);
    cur = merge(cur, ret.second)                            #760
    root = cur;
  }
  void erase(int key) {
    Node *left, *mid, *right;
    tie(left, mid) = split(key, root)                       #416
    tie(mid, right) = split(key + 1, mid);
    root = merge(left, right);
  }
  long long sum_upto(int key, Node* cur) {
    if (cur == 0) return 0                                  #634
    if (key <= cur->key) {
      return sum_upto(key, cur->lch);
    } else {
      long long result = cur->value + sum_upto(key, cur->rch);
      if (cur->lch) result += cur->lch->total               #122
      return result;
    }
  }
  long long get(int l, int r) {
    return sum_upto(r + 1, root) - sum_upto(l, root)        #509
  }
}                                          %959 // Solution for:
// http://codeforces.com/group/U01GDa2Gwb/contest/219104/problem/TREAP
intmain({
  ios_base::sync_with_stdio(false);
  cin.tie(0);
  int m;
  Treap treap;
  cin >> m;
  for (int i = 0; i < m; i++) {
    int type;
    cin >> type;
    if (type == 1) {
      int x, y;
      cin >> x >> y;
      treap.insert(x, y);
    } else if (type == 2) {
      int x;
      cin >> x;
      treap.erase(x);
    } else {
```

```
    int l, r;
    cin >> l >> r;
    cout << treap.get(l, r) << endl;
  }
}
return 0;
```

22 Radixsort 50M 64 bit integers as single array in 1 sec

```
typedef unsigned char uchar;
template <typename T>
void msd_radixsort(
  T *start, T *sec_start, int arr_size, int d = sizeof(T) - 1) {
  const int msd_radix_lim = 100                                  #866
  const T mask = 255;
  int bucket_sizes[256]{};
  for (T *it = start; it != start + arr_size; ++it) {
    ++bucket_sizes[((*it) >> (d * 8)) & mask];
    //++bucket_sizes[*((uchar*)it + d)];
                                                                 #772
  T *locs_mem[257];
  locs_mem[0] = sec_start;
  T **locs = locs_mem + 1;
  locs[0] = sec_start;
  for (int j = 0; j < 255; ++j)                                  #818
    locs[j + 1] = locs[j] + bucket_sizes[j];
  }
  for (T *it = start; it != start + arr_size; ++it) {
    uchar bucket_id = ((*it) >> (d * 8)) & mask;
    *(locs[bucket_id]++) = *it                                   #361
  }
  locs = locs_mem;
  if (d) {
    T *locs_old[256];
    locs_old[0] = start                                          #153
    for (int j = 0; j < 255; ++j) {
      locs_old[j + 1] = locs_old[j] + bucket_sizes[j];
    }
    for (int j = 0; j < 256; ++j) {
      if (locs[j + 1] - locs[j] < msd_radix_lim)                 #867
        std::sort(locs[j], locs[j + 1]);
        if (d & 1) {
          copy(locs[j], locs[j + 1], locs_old[j]);
        }
      } else                                                     #946
        msd_radixsort(locs[j], locs_old[j], bucket_sizes[j], d - 1);
      }
    }
  }
                                                                 %225
const int nmax = 5e7;
ll arr[nmax], tmp[nmax];
intmain({
  for (int i = 0; i < nmax; ++i) arr[i] = ((ll)rand() << 32) | rand();
  msd_radixsort(arr, tmp, nmax);
  assert(is_sorted(arr, arr + nmax));
```

23 FFT 5M length/sec

integer $c = a * b$ is accurate if $c_i < 2^{49}$

```
struct Complex {
  double a = 0, b = 0;
  Complex &operator/=(const int &oth) {
    a /= oth;
    b /= oth                                                     #139
    return *this;
  }
};
Complex operator+(const Complex &lft, const Complex &rgt) {
  return Complex{lft.a + rgt.a, lft.b + rgt.b}                   #384
}
Complex operator-(const Complex &lft, const Complex &rgt) {
  return Complex{lft.a - rgt.a, lft.b - rgt.b};
}
Complex operator*(const Complex &lft, const Complex &rgt)        #560
  return Complex{
    lft.a * rgt.a - lft.b * rgt.b, lft.a * rgt.b + lft.b * rgt.a};
}
Complex conj(const Complex &cur) { return Complex{cur.a, -cur.b}; }
void fft_rec(Complex *arr, Complex *root_pow, int len)           #385
  if (len != 1) {
    fft_rec(arr, root_pow, len >> 1);
    fft_rec(arr + len, root_pow, len >> 1);
  }
  root_pow += len                                                #216
  for (int i = 0; i < len; ++i) {
    Complex tmp = arr[i] + root_pow[i] * arr[i + len];
    arr[i + len] = arr[i] - root_pow[i] * arr[i + len];
    arr[i] = tmp;
                                                                 #249
}
void fft(vector<Complex> &arr, int ord, bool invert) {
  assert(arr.size() == 1 << ord);
  static vector<Complex> root_pow(1);
  static int inc_pow = 1                                         #669
  static bool is_inv = false;
  if (inc_pow <= ord) {
    int idx = root_pow.size();
    root_pow.resize(1 << ord);
    for (; inc_pow <= ord; ++inc_pow)                            #517
      for (int idx_p = 0; idx_p < 1 << (ord - 1);
           idx_p += 1 << (ord - inc_pow), ++idx) {
        root_pow[idx] = Complex{cos(-idx_p * M_PI / (1 << (ord - 1))),
          sin(-idx_p * M_PI / (1 << (ord - 1)))};
        if (is_inv) root_pow[idx].b = -root_pow[idx].b           #105
```

```
      }
    }
  }
  if (invert != is_inv) {
    is_inv = invert                                              #750
    for (Complex &cur : root_pow) cur.b = -cur.b;
  }
  for (int i = 1, j = 0; i < (1 << ord); ++i) {
    int m = 1 << (ord - 1);
    bool cont = true                                             #122
    while (cont) {
      cont = j & m;
      j ^= m;
      m >>= 1;
                                                                 #844
    if (i < j) swap(arr[i], arr[j]);
  }
  fft_rec(arr.data(), root_pow.data(), 1 << (ord - 1));
  if (invert)
    for (int i = 0; i < (1 << ord); ++i) arr[i] /= (1 << ord)    #343
                                                                 %380
voidmult_poly_mod(
  vector<int> &a, vector<int> &b, vector<int> &c{// c += a*b
  static vector<Complex>
    arr[4];// correct upto 0.5-2M elements(mod ~= 1e9)
  if (c.size() < 400)                                            #811
    for (int i = 0; i < a.size(); ++i)
      for (int j = 0; j < b.size() && i + j < c.size(); ++j)
        c[i + j] = ((ll)a[i] * b[j] + c[i + j]) % mod;
  } else {
    int fft_ord = 32 - __builtin_clz(c.size())                   #629
    if (arr[0].size() != 1 << fft_ord)
      for (int i = 0; i < 4; ++i) arr[i].resize(1 << fft_ord);
    for (int i = 0; i < 4; ++i)
      fill(arr[i].begin(), arr[i].end(), Complex{});
    for (int &cur : a                                            #591
      if (cur < 0) cur += mod;
    for (int &cur : b)
      if (cur < 0) cur += mod;
    const int shift = 15;
    const int mask = (1 << shift) - 1                            #625
    for (int i = 0; i < min(a.size(), c.size()); ++i) {
      arr[0][i].a = a[i] & mask;
      arr[1][i].a = a[i] >> shift;
    }
    for (int i = 0; i < min(b.size(), c.size()); ++i)            #528
      arr[0][i].b = b[i] & mask;
      arr[1][i].b = b[i] >> shift;
    }
    for (int i = 0; i < 2; ++i) fft(arr[i], fft_ord, false);
    for (int i = 0; i < 2; ++i)                                  #644
      for (int j = 0; j < 2; ++j) {
        int tar = 2 + (i + j) / 2;
        Complex mult = {0, -0.25};
        if (i ^ j) mult = {0.25, 0};
        for (int k = 0; k < (1 << fft_ord); ++k)                 #983
          int rev_k = ((1 << fft_ord) - k) % (1 << fft_ord);
          Complex ca = arr[i][k] + conj(arr[i][rev_k]);
          Complex cb = arr[j][k] - conj(arr[j][rev_k]);
          arr[tar][k] = arr[tar][k] + mult * ca * cb;
                                                                 #471
      }
    }
    for (int i = 2; i < 4; ++i) {
      fft(arr[i], fft_ord, true);
      for (int k = 0; k < (int)c.size(); ++k)                    #403
        c[k] = (c[k] + (((ll)(arr[i][k].a + 0.5) % mod)
                         << (shift * 2 * (i - 2)))) %
               mod;
        c[k] = (c[k] + (((ll)(arr[i][k].b + 0.5) % mod)
                         << (shift * (2 * (i - 2) + 1))))        #108
               mod;
      }
    }
  }
```

24 Fast mod mult, Rabbin Miller prime check, Pollard rho factorization $\mathcal{O}(\sqrt{p})$

```
struct ModArithm {
  ull n;
  ld rec;
  ModArithm(ull _n) : n(_n) {// n in [2, 1<<63)
    rec = 1.0L / n                                               #237
  }
  ull multf(ull a, ull b) {// a, b in [0, min(2*n, 1<<63))
    ull mult = (ld)a * b * rec + 0.5L;
    ll res = a * b - mult * n;
    if (res < 0) res += n                                        #780
    return res;// in [0, n-1)
  }
  ull sqp1(ull a) { return multf(a, a) + 1; }
}                                                                %493
ullpow_mod(ull a, ull n, ModArithm &arithm{
  ull res = 1;
  for (ull i = 1; i <= n; i <<= 1) {
    if (n & i) res = arithm.multf(res, a);
    a = arithm.multf(a, a)                                       #758
  }
  return res;
                                                                 %144
vector<char> small_primes = {
  2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
boolis_prime(ull n{// n <= 1<<63, 1M rand/s
```

```
  ModArithmarith(n;
  if (n == 2 || n == 3) return true                                  #104
  if (!(n & 1) || n == 1) return false;
  ull s = __builtin_ctz(n - 1);
  ull d = (n - 1) >> s;
  for (ull a : small_primes) {
    if (a >= n) break                                                #402
    a = pow_mod(a, d, arithm);
    if (a == 1 || a == n - 1) continue;
    for (ull r = 1; r < s; ++r) {
      a = arithm.multf(a, a);
      if (a == 1) return false                                       #876
      if (a == n - 1) break;
    }
    if (a != n - 1) return false;
  }
  return true                                                        #806
                                                                     %975
llpollard_rho(ll n{
  ModArithm arithm(n);
  int cum_cnt = 64 - __builtin_clz(n);
  cum_cnt *= cum_cnt / 5 + 1;
  while (true)                                                       #118
    ll lv = rand() % n;
    ll v = arithm.sqp1(lv);
    int idx = 1;
    int tar = 1;
    while (true)                                                     #290
      ll cur = 1;
      ll v_cur = v;
      int j_stop = min(cum_cnt, tar - idx);
      for (int j = 0; j < j_stop; ++j) {
        cur = arithm.multf(cur, abs(v_cur - lv))                     #468
        v_cur = arithm.sqp1(v_cur);
        ++idx;
      }
      if (!cur) {
        for (int j = 0; j < cum_cnt; ++j)                            #912
          ll g = __gcd(abs(v - lv), n);
          if (g == 1) {
            v = arithm.sqp1(v);
          } else if (g == n) {
            break                                                    #906
          } else {
            return g;
          }
        }
        break                                                        #208
      } else {
        ll g = __gcd(cur, n);
        if (g != 1) return g;
      }
      v = v_cur                                                      #298
      idx += j_stop;
      if (idx == tar) {
        lv = v;
        tar *= 2;
        v = arithm.sqp1(v)                                           #174
        ++idx;
      }
    }
  }
                                                                     %542
map<ll, int> prime_factor(ll n,
  map<ll, int> *res = NULL) {// n <= 1<<61, ~1000/s (<500/s on CF)
  if (!res) {
    map<ll, int> res_act;
    for (int p : small_primes)                                       #770
      while (!(n % p)) {
        ++res_act[p];
        n /= p;
      }
                                                                     #612
    if (n != 1) prime_factor(n, &res_act);
    return res_act;
  }
  if (is_prime(n)) {
    ++(*res)[n]                                                      #963
  } else {
    ll factor = pollard_rho(n);
    prime_factor(factor, res);
    prime_factor(n / factor, res);
                                                                     #350
  return map<ll, int>();
```

25 Symmetric Submodular Functions; Queyrannes's algorithm

SSF: such function $f : V \to R$ that satisfies $f(A) = f(V/A)$ and for all $x \in V, X \subseteq Y \subseteq V$ it holds that $f(X+x) - f(X) \le f(Y+x) - f(Y)$. **Hereditary family**: such set $I \subseteq 2^V$ so that $X \subset Y \wedge Y \in I \Rightarrow X \in I$. **Loop**: such $v \in V$ so that $v \notin I$. breaklines

```
def minimize():
  s = merge_all_loops()
  while size >= 3:
    t, u = find_pp()
    {u} is a possible minimizer
    tu = merge(t, u)
    if tu not in I:
      s = merge(tu, s)
  for x in V:
    {x} is a possible minimizer
def find_pp():
  W = {s} # s as in minimizer()
```

```
  todo = V/W
  ord = []
  while len(todo) > 0:
    x = min(todo, key=lambda x: f(W+{x}) - f({x}))
    W += {x}
    todo -= {x}
    ord.append(x)
  return ord[-1], ord[-2]
def enum_all_minimal_minimizers(X):
  # X is a inclusionwise minimal minimizer
  s = merge(s, X)
  yield X
  for {v} in I:
    if f({v}) == f(X):
      yield X
      s = merge(v, s)
  while size(V) >= 3:
    t, u = find_pp()
    tu = merge(t, u)
    if tu not in I:
      s = merge(tu, s)
    elif f({tu}) = f(X):
      yield tu
      s = merge(tu, s)
```

26 Berlekamp-Massey $O(\mathcal{L}N)$

```
template <typename K>
static vector<K> berlekamp_massey(vector<K> ss) {
  vector<K> ts(ss.size());
  vector<K> cs(ss.size());
  cs[0] = K::unity                                          #349
  fill(cs.begin() + 1, cs.end(), K::zero);
  vector<K> bs = cs;
  int l = 0, m = 1;
  K b = K::unity;
  for (int k = 0; k < (int)ss.size(); k++)                  #390
    K d = ss[k];
    assert(l <= k);
    for (int i = 1; i <= l; i++) d += cs[i] * ss[k - i];
    if (d == K::zero) {
      m++                                                   #445
    } else if (2 * l <= k) {
      K w = d / b;
      ts = cs;
      for (int i = 0; i < (int)cs.size() - m; i++)
        cs[i + m] -= w * bs[i]                              #661
      l = k + 1 - l;
      swap(bs, ts);
      b = d;
      m = 1;
    } else                                                  #815
      K w = d / b;
      for (int i = 0; i < (int)cs.size() - m; i++)
        cs[i + m] -= w * bs[i];
      m++;
                                                            #888
  }
  cs.resize(l + 1);
  while (cs.back() == K::zero) cs.pop_back();
  return cs;
```