University of Warsaw

ACM ICPC Team Reference Document

Wojciech Nadara
Marcin Smulewicz
Marek Sokołowski

```
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Headery.hpp                                                        %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include <bits/stdc++.h>

using namespace std;
#define PB push_back
#define MP make_pair
#define LL long long
#define int LL
#define FOR(i,a,b) for(int i = (a); i <= (b); i++)
#define RE(i,n) FOR(i,1,n)
#define REP(i,n) FOR(i,0,(int)(n)-1)
#define R(i,n) REP(i,n)
#define VI vector<int>
#define PII pair<int,int>
#define LD long double
#define FI first
#define SE second
#define st FI
#define nd SE
#define ALL(x) (x).begin(), (x).end()
#define SZ(x) ((int)(x).size())

template<class C> void mini(C &a4, C b4) { a4 = min(a4, b4); }
template<class C> void maxi(C &a4, C b4) { a4 = max(a4, b4); }

template<class TH> void _dbg(const char *sdbg, TH h){ cerr<<sdbg<<'='<<h<<endl; }
template<class TH, class... TA> void _dbg(const char *sdbg, TH h, TA... a) {
  while(*sdbg!=',')cerr<<*sdbg++;cerr<<'='<<h<<','; _dbg(sdbg+1, a...);
}

template<class T> ostream &operator<<(ostream& os, vector<T> V) {
  os << "["; for (auto vv : V) os << vv << ","; return os << "]";
}
template<class L, class R> ostream &operator<<(ostream &os, pair<L,R> P) {
  return os << "(" << P.st << "," << P.nd << ")";
}

#ifdef LOCAL
#define debug(...) _dbg(#__VA_ARGS__, __VA_ARGS__)
#else
#define debug(...) (__VA_ARGS__)
#define cerr if(0)cout
#endif

/*
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout << fixed << setprecision(11);
  cerr << fixed << setprecision(6);
}
*/


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      SETUP                                                              %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
:: GEANY:

Edit -> Preferences:
  Interface -> zmienic wszystkie czcionki na 8
  Editor -> Indentation -> Width = 2, Type = Spaces,
            Completions -> Autocompl. symbols, Autocompl. all words, num chars = 2

Tools -> Plugin Manager -> zaznaczyc Split Window -> OK


:: KOLORKI:
1) cp /etc/skel/.bashrc ~/
2) Odkomentowac force_color_prompt=yes
```



```
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Makefile                                                           %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
CXXFLAGS=-std=c++11 -g -fsanitize=address -D_GLIBCXX_DEBUG -DLOCAL -W -Wall -Wunused -W
shadow -Wuninitialized


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      2D(+,+).hpp                                                        %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
#define RO(a,aa) for(int a=aa;a;a-=(a&-a))
#define RX(a,aa) for(int a=aa;a<MAX;a+=(a&-a))
#define MAX 1010
LL dp1[MAX][MAX],dp2[MAX][MAX],dp3[MAX][MAX],dp4[MAX][MAX];
void add1(int xx,int yy,LL war){RO(x,xx)RO(y,yy)dp1[x][y]+=war;}
void add2(int xx,int yy,LL war){RO(x,xx)RX(y,yy)dp2[x][y]+=war;}
void add3(int xx,int yy,LL war){RX(x,xx)RO(y,yy)dp3[x][y]+=war;}
void add4(int xx,int yy,LL war){RX(x,xx)RX(y,yy)dp4[x][y]+=war;}
LL sum1(int xx,int yy){LL wyn=0;RX(x,xx)RX(y,yy)wyn+=dp1[x][y];return wyn;}
LL sum2(int xx,int yy){LL wyn=0;RX(x,xx)RO(y,yy)wyn+=dp2[x][y];return wyn;}
LL sum3(int xx,int yy){LL wyn=0;RO(x,xx)RX(y,yy)wyn+=dp3[x][y];return wyn;}
LL sum4(int xx,int yy){LL wyn=0;RO(x,xx)RO(y,yy)wyn+=dp4[x][y];return wyn;}
void addd(int x,int y,LL war){
  add1(x,y,war);add2(x,y,war*y);add3(x,y,war*x);add4(x,y,war*x*y);
}
LL summ(int x,int y){
  return sum1(x+1,y+1)*x*y + sum2(x+1,y)*x + sum3(x,y+1)*y + sum4(x,y);
}
void add(int x1,int y1,int x2,int y2,LL war){
  addd(x2,y2,war);addd(x2,y1,-war);addd(x1,y2,-war);addd(x1,y1,war);
}
LL sum(int x1,int y1,int x2,int y2){
  return summ(x2,y2)-summ(x1,y2)-summ(x2,y1)+summ(x1,y1);
}
int n,m;
int32_t main(){
  scanf("%lld%lld",&n,&m);
  while(m--){
    int z;
    scanf("%lld",&z);
    if(z==1){
      int a,b,c,d;
      scanf("%lld%lld%lld%lld",&a,&b,&c,&d);
      printf("%lld\n",sum(a,b,c+1,d+1));
    }else{
      int a,b,c,d;
      LL war;
      scanf("%lld%lld%lld%lld%lld",&a,&b,&c,&d,&war);
      add(a,b,c+1,d+1,war);
    }
  }
}
```

```
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Geo2D.h                                                   %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
const LD kEps = 1e-9;
const LD kPi = 2 * acos(0);
LD Sq(LD x) { return x * x; }
struct Point {
  LD x, y;
  Point() {}
  Point(LD a, LD b) : x(a), y(b) {}
  Point(const Point& a) : x(a.x), y(a.y) {}
  void operator=(const Point& a) { x = a.x; y = a.y; }
  Point operator+(const Point& a) const { Point p(x + a.x, y + a.y); return p; }
  Point operator-(const Point& a) const { Point p(x - a.x, y - a.y); return p; }
  Point operator*(LD a) const { Point p(x * a, y * a); return p; }
  Point operator/(LD a) const { assert(abs(a) > kEps); Point p(x / a, y / a); return p;
 }
  Point& operator+=(const Point& a) { x += a.x; y += a.y; return *this; }
  Point& operator-=(const Point& a) { x -= a.x; y -= a.y; return *this; }
  Point& operator*=(LD a) { x *= a; y *= a; return *this;}
  Point& operator/=(LD a) { assert(abs(a) > kEps); x /= a; y /= a; return *this; }

  bool IsZero() const { return abs(x) < kEps && abs(y) < kEps; }
  bool operator==(const Point& a) const { return (*this - a).IsZero(); }
  LD CrossProd(const Point& a) const { return x * a.y - y * a.x; }
  LD CrossProd(Point a, Point b) const { a -= *this; b -= *this; return a.CrossProd(b);
 }
  LD DotProd(const Point& a) const { return x * a.x + y * a.y; }
  LD Norm() const { return sqrt(Sq(x) + Sq(y)); }
  void NormalizeSelf() {  *this /= Norm(); }
  Point Normalize() { Point res(*this); res.NormalizeSelf(); return res; }
  LD Dist(const Point& a) const { return (*this - a).Norm(); }
  LD Angle() const { return atan2(y, x); }
  void RotateSelf(LD angle) {
    LD c = cos(angle); LD s = sin(angle);
    LD nx = x * c - y * s; LD ny = y * c + x * s;
    y = ny; x = nx;
  }
  Point Rotate(LD angle) { Point res(*this); res.RotateSelf(angle); return res; }
  static bool LexCmp(const Point& a, const Point& b) {
    if (abs(a.x - b.x) > kEps) { return a.x < b.x; }
    return a.y < b.y;
  }
  LD SqNorm() { return x * x + y * y; }
  friend ostream& operator<<(ostream& out, Point m);
};

ostream& operator<<(ostream& out, Point p) {
  out << "(" << p.x << ", " << p.y << ")"; return out;
}

struct Circle {
  Point center; LD r;
  Circle(LD x, LD y, LD rad) { center = Point(x, y); r = rad; }
  Circle(const Point& a, LD rad) : center(a), r(rad) {}
  LD Area() const { return kPi * Sq(r); }
  LD Perimeter() const { return 2 * kPi * r; }
  LD Diameter() const { return 2 * r; }
  Point RotateRightMost(LD ang) { return center + Point{r * cos(ang), r * sin(ang)}; }
  bool operator==(const Circle& c) { return center == c.center && abs(r - c.r) < kEps;
}
};

struct Line {
  Point p[2]; bool is_seg;
  Line(Point a, Point b, bool is_seg_ = false) {
    p[0] = a; p[1] = b; is_seg = is_seg_;
  }
  Point& operator[](int a) {
    return p[a];
  }
```

University of Warsaw Page 2

```
  Point NormalVector() {
    Point perp = p[1] - p[0];
    perp.RotateSelf(kPi / 2);
    perp.NormalizeSelf();
    return perp;
  }

  // (A, B, C) such that A^2 + B^2 = 1, (A, B) > (0, 0)
  vector<LD> LineEqNormLD() { // seems ok
    LD A = p[1].y - p[0].y; LD B = p[0].x - p[1].x;
    LD C = -(A * p[0].x + B * p[0].y);
    assert(abs(A * p[1].x + B * p[1].y + C) < kEps);
    LD norm = sqrt(Sq(A) + Sq(B));
    vector<LD> res{A, B, C}; for (auto& x : res) { x /= norm; }
    if (A < -kEps || (abs(A) < kEps && B < -kEps)) { for (auto& x : res) { x *= -1; } }
    return res;
  }

  // assumes that coordinates are integers!
  vector<int> LineEqNormInt() { // seems ok
    int A = round(p[1].y - p[0].y); int B = round(p[0].x - p[1].x);
    int C = -(A * p[0].x + B * p[0].y); int gcd = abs(__gcd(A, __gcd(B, C)));
    vector<int> res{A, B, C};
    for (auto& x : res) { x /= gcd; }
    if (A < 0 || (A == 0 && B < 0)) { for (auto& x : res) { x *= -1; } }
    return res;
  }
};

struct Utils {
  // 0, 1, 2 or 3 pts. In case of 3 pts it means they are equal
  static vector<Point> InterCircleCircle(Circle a, Circle b) {
    if (a.r + kEps < b.r) { swap(a, b); }
    if (a == b) {
      return vector<Point>{a.RotateRightMost(0), a.RotateRightMost(2 * kPi / 3),
          a.RotateRightMost(4 * kPi / 3)};
    }
    Point diff = b.center - a.center; LD dis = diff.Norm();  LD ang = diff.Angle();
    LD longest = max(max(a.r, b.r), dis); LD per = a.r + b.r + dis;
    if (2 * longest > per + kEps) { return vector<Point>(); }
    if (abs(2 * longest - per) < 2 * kEps) {
      return vector<Point>{a.RotateRightMost(ang)};
    }
    LD ang_dev = acos((Sq(a.r) + Sq(dis) - Sq(b.r)) / (2 * a.r * dis));
    return vector<Point>{a.RotateRightMost(ang - ang_dev), a.RotateRightMost(ang + ang_
dev)};
  }

  static vector<Point> InterLineLine(Line& a, Line& b) { // working fine
    Point vec_a = a[1] - a[0]; Point vec_b1 = b[1] - a[0]; Point vec_b0 = b[0] - a[0];
    LD tr_area = vec_b1.CrossProd(vec_b0);
    LD quad_area = vec_b1.CrossProd(vec_a) + vec_a.CrossProd(vec_b0);
    if (abs(quad_area) < kEps) { // parallel or coinciding
      if (PtBelongToLine(b, a[0])) { return {a[0], a[1]}; }
      else { return {}; }
    }
    return {a[0] + vec_a * (tr_area / quad_area)};
  }

  static Point ProjPointToLine(Point p, Line l) { ///Tested
    Point diff = l[1] - l[0];
    return l[0] + diff * (diff.DotProd(p - l[0]) / diff.DotProd(diff));
  }

  static Point ReflectPtWRTLine(Point p, Line l) {
    Point proj = ProjPointToLine(p, l); return proj * 2 - p;
  }

  static vector<Point> InterCircleLine(Circle c, Line l) {
    Point proj = ProjPointToLine(c.center, l); LD dis_proj = c.center.Dist(proj);
    if (dis_proj > c.r + kEps) { return vector<Point>(); }
    if (dis_proj > c.r - kEps) { return {proj}; }
```

```
    LD a = sqrt(Sq(c.r)-Sq(dis_proj)); Point dir = l[1]-l[0]; LD dir_norm = dir.Norm();
    vector<Point> cands{proj + dir * (a / dir_norm), proj - dir * (a / dir_norm)};
    if (cands[0].Dist(cands[1]) < kEps) { return vector<Point>{proj}; }
    return cands;
  }

  static bool PtBelongToLine(Line l, Point p) { return abs(l[0].CrossProd(l[1], p)) < k
Eps; }

  static bool PtBelongToSeg(Line l, Point p) { // seems ok
    return abs(p.Dist(l[0]) + p.Dist(l[1]) - l[0].Dist(l[1])) < kEps;
  }

  static vector<Point> InterCircleSeg(Circle c, Line l) { //seems ok
    vector<Point> from_line = InterCircleLine(c, l);
    vector<Point> res;
    for (auto p : from_line) {
      if (PtBelongToSeg(l, p)) { res.PB(p); }
    }
    return res;
  }

  static vector<Point> TangencyPtsToCircle(Circle c, Point p) { // seems ok
    LD d = c.center.Dist(p); if (d < c.r - kEps) { return {}; }
    if (d < c.r + kEps) { return {p}; }
    LD from_cent = (p - c.center).Angle(); LD ang_dev = acos(c.r / d);
    return {c.RotateRightMost(from_cent - ang_dev), c.RotateRightMost(from_cent + ang_d
ev)};
  }

  // outer and inner tangents tested only locally (however I believe that rigorously)
  static vector<Line> OuterTangents(Circle c1, Circle c2) {
    if (c1 == c2) { return {}; } // is it surely best choice?
    if (c1.r < c2.r) { swap(c1, c2); }
    if (c2.r + c1.center.Dist(c2.center) < c1.r - kEps) { return {}; }
    if (abs(c1.r - c2.r) < kEps) {
      Point diff = c2.center - c1.center;
      Point R = diff.Rotate(kPi / 2) * (c1.r / diff.Norm());
      return {{c1.center + R, c2.center + R}, {c1.center - R, c2.center - R}};
    }
    Point I = c1.center + (c2.center - c1.center) * (c1.r / (c1.r - c2.r));
    if (c2.r + c1.center.Dist(c2.center) < c1.r + kEps) {
      return {{I, I + (c2.center - c1.center).Rotate(kPi / 2)}};
    }
    vector<Point> to1 = TangencyPtsToCircle(c1, I);
    vector<Point> to2 = TangencyPtsToCircle(c2, I);
    vector<Line> res{{to1[0], to2[0]}, {to1[1], to2[1]}};
    assert(Utils::PtBelongToLine(res[0], I));
    assert(Utils::PtBelongToLine(res[1], I));
    return res;
  }

  static vector<Line> InnerTangents(Circle c1, Circle c2) {
    if (c1 == c2) { return {}; } // this time surely best choice
    if (c1.r < c2.r) { swap(c1, c2); }
    LD d = c1.center.Dist(c2.center);
    if (d < c1.r + c2.r - kEps) { return {}; }
    Point I = c1.center + (c2.center - c1.center) * (c1.r / (c1.r + c2.r));
    if (d < c1.r + c2.r + kEps) {
      return {{I, I + (c2.center - c1.center).Rotate(kPi / 2)}};
    }
    vector<Point> to1 = TangencyPtsToCircle(c1, I);
    vector<Point> to2 = TangencyPtsToCircle(c2, I);
    vector<Line> res{{to1[0], to2[0]}, {to1[1], to2[1]}};
    assert(Utils::PtBelongToLine(res[0], I));
    assert(Utils::PtBelongToLine(res[1], I));
    return res;
  }

  static bool AreParallel(Line l1, Line l2) { // seems ok
    return abs(l1[0].CrossProd(l2[0], l1[1]) - l1[0].CrossProd(l2[1], l1[1])) < kEps;
  }
```



```
  static vector<Point> InterSegs(Line l1, Line l2) { // seems ok
    if (!Point::LexCmp(l1[0], l1[1])) { swap(l1[0], l1[1]); }
    if (!Point::LexCmp(l2[0], l2[1])) { swap(l2[0], l2[1]); }
    if (AreParallel(l1, l2)) {
      if (!PtBelongToLine(l1, l2[0])) { return vector<Point>(); }
      vector<Point> ends(2);
      for (int tr = 0; tr < 2; tr++) {
        if (Point::LexCmp(l1[tr], l2[tr]) ^ tr) { ends[tr] = l2[tr]; }
        else { ends[tr] = l1[tr]; }
      }
      if ((ends[1] - ends[0]).IsZero()) { ends.pop_back(); }
      if (SZ(ends) == 2 && Point::LexCmp(ends[1], ends[0])) { return vector<Point>(); }
      return ends;
    } else {
      vector<Point> p = InterLineLine(l1, l2);
      if (PtBelongToSeg(l1, p[0]) && PtBelongToSeg(l2, p[0])) { return p; }
      return vector<Point>();
    }
  }

  static LD Angle(Point P, Point Q, Point R) { // angle PQR
    LD ang2 = (P - Q).Angle(); LD ang1 = (R - Q).Angle();
    LD ans = ang1 - ang2;
    if (ans < kEps) { ans += 2 * kPi; }
    return ans;
  }

  static LD DiskInterArea(Circle c1, Circle c2) {
    if (c1.r < c2.r) { swap(c1, c2); }
    LD d = c1.center.Dist(c2.center);
    if (c1.r + c2.r < d + kEps) { return 0; }
    if (c1.r - c2.r > d - kEps) { return kPi * Sq(c2.r); }
    LD alfa = acos((Sq(d) + Sq(c1.r) - Sq(c2.r)) / (2 * d * c1.r));
    LD beta = acos((Sq(d) + Sq(c2.r) - Sq(c1.r)) / (2 * d * c2.r));
    return alfa * Sq(c1.r) + beta * Sq(c2.r)
        - sin(2 * alfa) * Sq(c1.r) / 2 - sin(2 * beta) * Sq(c2.r) / 2;
  }

  static Line RadAxis(Circle c1, Circle c2) {
    LD d = c1.center.Dist(c2.center); LD a = (Sq(c1.r) - Sq(c2.r) + Sq(d)) / (2 * d);
    Point Q = c1.center + (c2.center - c1.center) * (a / d);
    Point R = Q + (c2.center - c1.center).Rotate(kPi / 2);
    return Line(Q, R);
  }
};

struct Polygon {
  vector<Point> pts;
  void Add(Point p) { pts.push_back(p); }
  double Area() { // positive for counterclockwise
    double area = 0;
    for (int i = 0; i < SZ(pts); i++) {
      area += pts[i].CrossProd(pts[(i + 1) % SZ(pts)]);
    }
    return area / 2;
  }
  void OrientCounterclockwise() {
    if (Area() < 0) { reverse(pts.begin(), pts.end()); }
  }
  int next(int a) {
    if (a + 1 < SZ(pts)) { return a + 1; }
    return 0;
  }
  pair<int, int> FurthestPair() {
    MakeConvexHull();
    OrientCounterclockwise();
    int furth = 1;
    pair<int, int> best_pair = make_pair(0, 0);
    double best_dis = 0;
    for (int i = 0; i < SZ(pts); i++) {
      Point side = pts[next(i)] - pts[i];
```

```
      while (side.CrossProd(pts[furth] - pts[i]) <
             side.CrossProd(pts[next(furth)] - pts[i])) {
        furth = next(furth);
      }
      vector<int> vec{i, next(i)};
      for (auto ind : vec) {
        if (pts[ind].Dist(pts[furth]) > best_dis) {
          best_pair = make_pair(ind, furth); best_dis = pts[ind].Dist(pts[furth]);
        }
      }
      cerr<<"Furthest from: "<<pts[i]<<"-"<<pts[next(i)]<<" is "<<pts[furth]<<endl;
    }
    return best_pair;
  }
  // for square 34
  //            12 holds one_way_hull = {{1,3,4},{1,2,4}}
  void MakeConvexHull() {
    vector<Point> one_way_hull[2];
    sort(pts.begin(), pts.end(), Point::LexCmp);
    for (int dir = -1; dir <= 1; dir += 2) {
      int hull_num = (dir + 1) / 2;
      auto& H = one_way_hull[hull_num];
      one_way_hull[hull_num].push_back(pts[0]);
      if (SZ(pts) > 1) { H.push_back(pts[1]); }
      for (int i = 2; i < SZ(pts); i++) {
        while (SZ(H) >= 2 &&
            dir * (pts[i] - H[SZ(H) - 2]).CrossProd(H.back() - H[SZ(H) - 2]) > -kEps) {
          H.pop_back();
        }
        H.push_back(pts[i]);
      }
    }
    pts.clear();
    for (auto p : one_way_hull[1]) { pts.push_back(p); }
    for (int i = SZ(one_way_hull[0]) - 2; i >= 1; i--) {
      pts.push_back(one_way_hull[0][i]);
    }
  }
  // without sides
  vector<vector<bool>> InsideDiagonalsMatrix() {
    int n = pts.size();
    vector<vector<bool>> res(n, vector<bool>(n));
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        Line diag(pts[i], pts[j]);
        if (i == j || abs(i - j) == 1 || abs(i - j) == n - 1) { continue; }
        res[i][j] = 1;
        for (int k = 0; k < n; k++) {
          int kk = next(k);
          Line side(pts[k], pts[kk]);
          if (k == i || k == j || kk == i || kk == j) { continue; }
          vector<Point> inter = Utils::InterSegs(diag, side);
          if (SZ(inter)) { res[i][j] = 0; }
        }
        int act = next(i); LD areas[2] = {0, 0}; int passed_j = 0;
        while (act != i) {
          passed_j |= (act == j);
          areas[passed_j] += pts[i].CrossProd(pts[act], pts[next(act)]);
          act = next(act);
        }
        if (areas[0] * areas[1] < kEps) { res[i][j] = 0; }
      }
    }
    return res;
  }
  // P needs to be strictly outside polygon
  // polygon needs to be STRICTLY convex and counterclockwise oriented (as MakeConvexHu
ll does)
  // returns {L, R} so that PL, PR are tangents and PL is on left
  vector<Point> Tangents(Point p) { // tested here: https://icpc.kattis.com/problems/sp
in (1169964)
    vector<Point> res;
```



```
    REP (tr, 2) {
      auto GrThan = [&](int fir, int sec) { // fir on sec's left
        return p.CrossProd(pts[sec], pts[fir]) > kEps;
      };
      bool up = false; int cr = 0;
      if (SZ(pts) >= 2) { cr = p.CrossProd(pts[0], pts[1]); }
      if (abs(cr) < kEps && SZ(pts) >= 3) { cr = p.CrossProd(pts[0], pts[2]); }
      up = (cr > 0);
      VI bd{1, SZ(pts) - 1};
      int faj = 0;
      while (bd[0] + 6 <= bd[1]) { // better don't replace with smaller constants
        VI h(2);
        REP (hh, 2) { h[hh] = (bd[0] + bd[1] + bd[hh]) / 3; }
        if (!GrThan(h[up ^ tr], 0) ^ tr) { bd[up ^ tr] = h[up ^ tr]; }
        else {
          int gr = GrThan(h[0], h[1]);
          bd[gr ^ tr] = h[gr ^ tr];
        }
      }
      FOR (i, bd[0], bd[1]) {
        if (GrThan(i, faj) ^ tr) { faj = i; }
      }
      res.PB(pts[faj]);
    }
    return res;
  }
};

struct ConvexPolHalves { // tested here: https://icpc.kattis.com/problems/spin (1169964
)
  vector<vector<Point>> chains; // initialized by MakeConvexHull
  bool BelongTo(Point p) { // including borders
    if (SZ(chains[0]) == 1) {
      return (chains[0][0] - p).IsZero();
    }
    if (p.x  + kEps < chains[0][0].x || p.x - kEps > chains[0].back().x) { return false
; }
    REP (tr, 2) {
      int kl = 0, kp = SZ(chains[tr]) - 2, faj = 0;
      while (kl <= kp) {
        int aktc = (kl + kp) / 2;
        if (chains[tr][aktc].x < p.x + kEps) {
          kl = aktc + 1;
          faj = aktc;
        } else {
          kp = aktc - 1;
        }
      }
      Point fir = chains[tr][faj], sec = chains[tr][faj + 1];
      if (abs(fir.x - sec.x) < kEps) {
        if (tr == 0) { if (sec.y + kEps < p.y) { return false; } }
        else { if (fir.y - kEps > p.y) { return false; } }
      } else {
        LD cr = fir.CrossProd(sec, p);
        if (abs(cr) < kEps) { return true; }
        if ((cr > 0) ^ tr) { return false; }
      }
    }
    return true;
  }
};

// CLIP START
bool InUpper(Point a) {
  if (abs(a.y) > kEps) { return a.y > 0; } return a.x > 0;
}
bool angle_cmp(const Point a, const Point b) {
  bool u = InUpper(a); bool v = InUpper(b);
  return u!=v ? u : a.CrossProd(b)>0;
}
/** @brief a+(b-a)*f \in c+lin(d-c)  @returns f */
LD cross(Point a, Point b, Point c, Point d) {
```

```
  return (d - c).CrossProd(a - c) / (d - c).CrossProd(a - b);
}

struct ClipLine { // valid side is on left
  ClipLine(Point A, Point B) : al(A), bl(B), a(A), b(B) {};
  Point al,bl; // original line points
  mutable Point a,b; // actual intersection points
  Point dir() const { return bl - al; }
  bool operator<(const ClipLine& l) const { return angle_cmp(dir(),l.dir()); }
  Point cross(const ClipLine& l) {
    return al + (bl - al) * ::cross(al, bl, l.al, l.bl);
  }
  bool left(Point p) { return (bl - al).CrossProd(p - al) > 0; }
};

struct Clip {
  Clip(LD r) : area(4*r*r) {
    Point a{-r,-r}, b{r,-r}, c{r,r}, d{-r,r};
    lines = {ClipLine(a,b), ClipLine(b,c), ClipLine(c,d), ClipLine(d,a)};
  }
  // doesn't work when two equal lines are inserted
  // in such case create set of normalized equations of lines with custom == kEps
  void insert(Line l) { insert(ClipLine(l[0], l[1])); }
  void insert(ClipLine l) {
    assert(abs(l.dir().SqNorm()) > kEps); find(l);
    while (size() && !l.left(it->a) && !l.left(it->b)) { erase(); }
    if (size()) {
      while (prev(), size() && !l.left(it->a) && !l.left(it->b)) { erase(); }
    }
    if (size() && (!l.left(it->a) || !l.left(it->b))) {
      l.a = l.cross(*it);
      area -= l.a.CrossProd(it->b)*.5; it->b = l.a; next();
      l.b = l.cross(*it);
      if ((l.a-l.b).SqNorm() < kEps) { l.b = l.a; }
      area -= it->a.CrossProd(l.b) * .5;
      it->a = l.b;
      if (!(l.a - l.b).IsZero()) { area += l.a.CrossProd(l.b)*.5; lines.insert(l); }
    }
  }
  void find(const ClipLine &l) {
    it = lines.lower_bound(l); if (it == lines.end()) { it = lines.begin(); }
  }
  void recalculate() {
    area = 0; for (const ClipLine &l : lines) area += l.a.CrossProd(l.b); area *= .5;
  }
  int size() { return lines.size(); }
  void next() { if(++it==lines.end()) it = lines.begin(); }
  void prev() { if(it==lines.begin()) it = lines.end(); --it; }
  void erase() {
    assert(it!=lines.end()); area -= it->a.CrossProd(it->b)*.5; it = lines.erase(it);
    if(it==lines.end()) it = lines.begin();
  }
  typename set<ClipLine>::iterator it; set<ClipLine> lines; LD area;
};
// CLIP ENDS

// CENTERS BEGIN
Point Bary(Point A, Point B, Point C, LD a, LD b, LD c) {
  return (A * a + B * b + C * c) / (a + b + c);
}
Point Centroid(Point A, Point B, Point C) { return Bary(A, B, C, 1, 1, 1); }
Point Circumcenter(Point A, Point B, Point C) {
  LD a = (B - C).SqNorm(), b = (C - A).SqNorm(), c = (A - B).SqNorm();
  return Bary(A, B, C, a * (b + c - a), b * (c + a - b), c * (a + b - c));
}
Point Incenter(Point A, Point B, Point C) {
  return Bary(A, B, C, (B - C).Norm(), (A - C).Norm(), (A - B).Norm());
}
Point Orthocenter(Point A, Point B, Point C) {
  LD a = (B - C).SqNorm(), b = (C - A).SqNorm(), c = (A - B).SqNorm();
  return Bary(A, B, C, (a+b-c)*(c+a-b), (b+c-a)*(a+b-c), (c+a-b)*(b+c-a));
}
```

University of Warsaw Page 5

```
Point Excenter(Point A, Point B, Point C) { // opposite to A
  LD a = (B - C).Norm(), b = (A - C).Norm(), c = (A - B).Norm();
  return Bary(A, B, C, -a, b, c);
}


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Geo3D.h                                                              %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Geo2D.h"
struct Point3 {
  LD x, y, z;
  Point3 operator+(Point3 a) { Point3 p{x + a.x, y + a.y, z + a.z}; return p; }
  Point3 operator-(Point3 a) { Point3 p{x - a.x, y - a.y, z - a.z}; return p; }
  Point3 operator*(LD a) { Point3 p{x * a, y * a, z * a}; return p; }
  Point3 operator/(LD a) { assert(a > kEps); Point3 p{x / a, y / a, z / a}; return p; }
  Point3& operator+=(Point3 a) { x += a.x; y += a.y; z += a.z; return *this; }
  Point3& operator-=(Point3 a) { x -= a.x; y -= a.y; z -= a.z; return *this; }
  Point3& operator*=(LD a) { x *= a; y *= a; z *= a; return *this;}
  Point3& operator/=(LD a) { assert(a > kEps); x /= a; y /= a; z /= a; return *this; }

  LD& operator[](int a) {
    if (a == 0) { return x; } if (a == 1) { return y; } return z;
  }
  bool IsZero() { return abs(x) < kEps && abs(y) < kEps && abs(z) < kEps; }
  LD DotProd(Point3 a) { return x * a.x + y * a.y + z * a.z; }
  LD Norm() { return sqrt(x * x + y * y + z * z); }
  LD SqNorm() { return x * x + y * y + z * z; }
  void NormalizeSelf() { *this /= Norm(); }
  Point3 Normalize() {
    Point3 res(*this); res.NormalizeSelf(); return res;
  }
  LD Dis(Point3 a) { return (*this - a).Norm(); }
  pair<LD, LD> SphericalAngles() { return {atan2(z, sqrt(x * x + y * y)), atan2(y, x)};
 }
  LD Area(Point3 p) { return Norm() * p.Norm() * sin(Angle(p)) / 2; }
  LD Angle(Point3 p) {
    LD a = Norm(); LD b = p.Norm(); LD c = Dis(p);
    return acos((a * a + b * b - c * c) / (2 * a * b));
  }
  static LD Angle(Point3 p, Point3 q) { return p.Angle(q); }
  Point3 CrossProd(Point3 p) {
    Point3 q(*this);
    return {q[1] * p[2] - q[2] * p[1],
            q[2] * p[0] - q[0] * p[2],
            q[0] * p[1] - q[1] * p[0]};
  }
  static bool LexCmp(Point3& a, const Point3& b) {
    if (abs(a.x - b.x) > kEps) { return a.x < b.x; }
    if (abs(a.y - b.y) > kEps) { return a.y < b.y; }
    return a.z < b.z;
  }
  friend ostream& operator<<(ostream& out, Point3 m);
};
ostream& operator<<(ostream& out, Point3 p) {
  out << "(" << p.x << ", " << p.y << ", " << p.z << ")"; return out;
}

struct Line3 {
  Point3 p[2]; Point3& operator[](int a) { return p[a]; }
  friend ostream& operator<<(ostream& out, Line3 m);
};
ostream& operator<<(ostream& out, Line3 l) {
  out << l[0] << " - " << l[1]; return out;
}

struct Plane {
  Point3 p[3];
  Point3& operator[](int a) { return p[a]; }
  Point3 GetNormal() {
    Point3 cross = (p[1] - p[0]).CrossProd(p[2] - p[0]); return cross.Normalize();
  }
```

```
  void GetPlaneEq(LD& A, LD& B, LD& C, LD& D) {
    Point3 normal = GetNormal();
    A = normal[0]; B = normal[1]; C = normal[2];
    D = normal.DotProd(p[0]);
    assert(abs(D - normal.DotProd(p[1])) < kEps);
    assert(abs(D - normal.DotProd(p[2])) < kEps);
  }
  vector<Point3> GetOrtonormalBase() {
    Point3 normal = GetNormal();
    Point3 cand = {-normal.y, normal.x, 0};
    if (abs(cand.x) < kEps && abs(cand.y) < kEps) { cand = {0, -normal.z, normal.y}; }
    cand.NormalizeSelf();
    Point3 third = Plane{Point3{0, 0, 0}, normal, cand}.GetNormal();
    assert(abs(normal.DotProd(cand)) < kEps &&
           abs(normal.DotProd(third)) < kEps && abs(cand.DotProd(third)) < kEps);
    return {normal, cand, third};
  }
};

struct Circle3 {
  Plane pl; Point3 o; LD r;
  friend ostream& operator<<(ostream& out, Circle3 m);
};

ostream& operator<<(ostream& out, Circle3 c) {
  out << "pl: (" << c.pl[0] << ", " << c.pl[1] << ", " << c.pl[2] << "), cent: " << c.o
 << " r: " << c.r << "\n";
  return out;
}

struct Sphere {
  Point3 cent; LD r;
};

struct Utils3 {
  static bool Lines3Equal(Line3 p, Line3 l) {
    return Utils3::PtBelongToLine3(p[0], l) && Utils3::PtBelongToLine3(p[1], l);
  }

  //angle PQR
  static LD Angle(Point3 P, Point3 Q, Point3 R) { return (P - Q).Angle(R - Q); }

  static Point3 ProjPtToLine3(Point3 p, Line3 l) { // ok
    Point3 diff = l[1] - l[0]; diff.NormalizeSelf();
    return l[0] + diff * (p - l[0]).DotProd(diff);
  }

  static LD DisPtLine3(Point3 p, Line3 l) { // ok
//  LD area = Area(p, l[0], l[1]); LD dis1 = 2 * area / l[0].Dis(l[1]);
    LD dis2 = p.Dis(ProjPtToLine3(p, l)); // assert(abs(dis1 - dis2) < kEps);
    return dis2;
  }

  static LD DisPtPlane(Point3 p, Plane pl) {
    Point3 normal = pl.GetNormal(); return abs(normal.DotProd(p - pl[0]));
  }

  static Point3 ProjPtToPlane(Point3 p, Plane pl) {
    Point3 normal = pl.GetNormal(); return p - normal * normal.DotProd(p - pl[0]);
  }

  static bool PtBelongToPlane(Point3 p, Plane pl) { return DisPtPlane(p, pl) < kEps; }

  static Point PlanePtTo2D(Plane pl, Point3 p) { // ok
    assert(PtBelongToPlane(p, pl));
    vector<Point3> base = pl.GetOrtonormalBase();
    Point3 control{0, 0, 0};
    REP (tr, 3) { control += base[tr] * p.DotProd(base[tr]); }
    assert(PtBelongToPlane(pl[0] + base[1], pl));
    assert(PtBelongToPlane(pl[0] + base[2], pl));
    assert((p - control).IsZero());
    return {p.DotProd(base[1]), p.DotProd(base[2])};
```



```
  }

  static Line PlaneLineTo2D(Plane pl, Line3 l) {
    return {PlanePtTo2D(pl, l[0]), PlanePtTo2D(pl, l[1])};
  }

  static Point3 PlanePtTo3D(Plane pl, Point p) { // ok
    vector<Point3> base = pl.GetOrtonormalBase();
    return base[0] * base[0].DotProd(pl[0]) + base[1] * p.x + base[2] * p.y;
  }

  static Line3 PlaneLineTo3D(Plane pl, Line l) {
    return {PlanePtTo3D(pl, l[0]), PlanePtTo3D(pl, l[1])};
  }

  static Line3 ProjLineToPlane(Line3 l, Plane pl) { // ok
    return {ProjPtToPlane(l[0], pl), ProjPtToPlane(l[1], pl)};
  }
  static LD DisLineLine(Line3 l, Line3 k) { // ok
    Plane together {l[0], l[1], l[0] + k[1] - k[0]}; // parallel FIXME
    Line3 proj = ProjLineToPlane(k, together);
    Point3 inter = (Utils3::InterLineLine(l, proj))[0];
    Point3 on_k_inter = k[0] + inter - proj[0];
    return inter.Dis(on_k_inter);
  }

  static bool PtBelongToLine3(Point3 p, Line3 l) { return DisPtLine3(p, l) < kEps; }

  static bool Line3BelongToPlane(Line3 l, Plane pl) {
    return PtBelongToPlane(l[0], pl) && PtBelongToPlane(l[1], pl);
  }

  static LD Det(Point3 a, Point3 b, Point3 d) { // ok
    Point3 pts[3] = {a, b, d};
    LD res = 0;
    for (int sign : {-1, 1}) {
      REP (st_col, 3) {
        int c = st_col;
        LD prod = 1;
        REP (r, 3) {
          prod *= pts[r][c];
          c = (c + sign + 3) % 3;
        }
        res += sign * prod;
      }
    }
    return res;
  }

  static LD Area(Point3 p, Point3 q, Point3 r) { q -= p; r -= p; return q.Area(r); }

  static vector<Point3> InterLineLine(Line3 k, Line3 l) {
    if (Lines3Equal(k, l)) { return {k[0], k[1]}; }
    if (PtBelongToLine3(l[0], k)) { return {l[0]}; }
    Plane pl{l[0], k[0], k[1]};
    if (!PtBelongToPlane(l[1], pl)) { return {}; }
    Line k2 = PlaneLineTo2D(pl, k); Line l2 = PlaneLineTo2D(pl, l);
    vector<Point> inter = Utils::InterLineLine(k2, l2);
    vector<Point3> res;
    for (auto P : inter) { res.PB(PlanePtTo3D(pl, P)); }
    return res;
  }

  static Plane ParallelPlane(Plane pl, Point3 A) { // plane parallel to pl going throug
h A
    Point3 diff = A - ProjPtToPlane(A, pl);
    return {pl[0] + diff, pl[1] + diff, pl[2] + diff};
  }

  // image of B in rotation wrt line passing through origin s.t. A1->A2
  // implemented in more general case with similarity instead of rotation
  static Point3 RotateAccordingly(Point3 A1, Point3 A2, Point3 B1) { // ok
```

```
    Plane pl{A1, A2, {0, 0, 0}};
    Point A12 = PlanePtTo2D(pl, A1); Point A22 = PlanePtTo2D(pl, A2);
    complex<LD> rat = complex<LD>(A22.x, A22.y) / complex<LD>(A12.x, A12.y);
    Plane plb = ParallelPlane(pl, B1); Point B2 = PlanePtTo2D(plb, B1);
    complex<LD> Brot = rat * complex<LD>(B2.x, B2.y);
    return PlanePtTo3D(plb, {Brot.real(), Brot.imag()});
  }

  static vector<Circle3> InterSpherePlane(Sphere s, Plane pl) { // ok
    Point3 proj = ProjPtToPlane(s.o, pl);
    LD dis = s.o.Dis(proj);
    if (dis > s.r + kEps) { return {}; }
    if (dis > s.r - kEps) { return {{pl, proj, 0}}; } // is it best choice?
    return {{pl, proj, sqrt(s.r * s.r - dis * dis)}};
  }

  static bool PtBelongToSphere(Sphere s, Point3 p) {
    return abs(s.r - s.o.Dis(p)) < kEps;
  }
};

struct PointS { // just for conversion purposes, probably toEucl suffices
  LD lat, lon;
  Point3 toEucl() {
    return Point3{cos(lat) * cos(lon), cos(lat) * sin(lon), sin(lat)};
  }
  PointS(Point3 p) {
    p.NormalizeSelf(); lat = asin(p.z); lon = acos(p.y / cos(lat));
  }
};

LD DistS(Point3 a, Point3 b) {
  return atan2l(b.CrossProd(a).Norm(), a.DotProd(b));
}

struct CircleS {
  Point3 o; // center of circle on sphere
  LD r; // arc len
  LD area() const { return 2 * kPi * (1 - cos(r)); }
};

CircleS From3(Point3 a, Point3 b, Point3 c) { // any three different points
  int tmp = 1;
  if ((a - b).Norm() > (c - b).Norm()) { swap(a, c); tmp = -tmp; }
  if ((b - c).Norm() > (a - c).Norm()) { swap(a, b); tmp = -tmp; }
  Point3 v = (c - b).CrossProd(b - a); v = v * (tmp / v.Norm());
  return CircleS{v, DistS(a,v)};
}

CircleS From2(Point3 a, Point3 b) { // neither the same nor the opposite
  Point3 mid = (a + b) / 2; mid = mid / mid.Norm(); return From3(a, mid, b);
}

LD SphAngle(Point3 A, Point3 B, Point3 C) { //angle at A, no two points opposite
  LD a = B.DotProd(C); LD b = C.DotProd(A); LD c = A.DotProd(A);
  return acos((b - a * c) / sqrt((1 - Sq(a)) * (1 - Sq(c))));
}

LD TriangleArea(Point3 A, Point3 B, Point3 C) { // no two poins opposite
  LD a = SphAngle(C, A, B); LD b = SphAngle(A, B, C); LD c = SphAngle(B, C, A);
  return a + b + c - kPi;
}

vector<Point3> IntersectionS(CircleS c1, CircleS c2) {
  Point3 n = c2.o.CrossProd(c1.o), w = c2.o * cos(c1.r) - c1.o * cos(c2.r);
  LD d = n.SqNorm();
  if (d < kEps) { return {}; } // parallel circles (can fully overlap)
  LD a = w.SqNorm() / d;
  vector<Point3> res;
  if (a >= 1 + kEps) { return res; }
  Point3 u = n.CrossProd(w) / d;
  if (a > 1 - kEps) { res.PB(u); return res; }
```

University of Warsaw Page 7

```
  LD h = sqrt((1 - a) / d);
  res.PB(u + n * h); res.PB(u - n * h);
  return res;
}

bool Eq(LD a, LD b) { return abs(a - b) < kEps; }

vector<Point3> intersect(Sphere a, Sphere b, Sphere c) { // Does not work for 3 colinea
r centers
    vector<Point3> res;
    Point3 ex, ey, ez;
    LD r1 = a.r, r2 = b.r, r3 = c.r, d, cnd_x = 0, i, j;
    ex = (b.o - a.o).Normalize();
    i = ex.DotProd(c.o - a.o);
    ey = ((c.o - a.o) - ex * i).Normalize();
    ez = ex.CrossProd(ey);
    d = (b.o - a.o).Norm();
    j = ey.DotProd(c.o - a.o);

    bool cnd = 0;
    if (Eq(r2, d - r1)) { cnd_x = +r1; cnd = 1; }
    if (Eq(r2, d + r1)) { cnd_x = -r1; cnd = 1; }

    if (!cnd && (r2 < d - r1 || r2 > d + r1)) return res;
    if (cnd) {
      if (Eq(Sq(r3), (Sq(cnd_x - i) + Sq(j)))) res.PB(Point3(cnd_x, 0.0, 0.0));
    } else {
      LD x = (Sq(r1) - Sq(r2) + Sq(d)) / (2 * d);
      LD y = (Sq(r1) - Sq(r3) + Sq(i) + Sq(j)) / (2 * j) - (i / j) * x;
      LD u = Sq(r1) - Sq(x) - Sq(y);
      if (u >= -kEps) {
        LD z = sqrtl(max(LD(0), u)); res.PB(Pt(x, y, z));
        if (!isZero(z)) res.PB(Pt(x, y, -z));
      }
    }
    for (auto& it : res) { it = a.o + ex * it[0] + ey * it[1] + ez * it[2]; }
    return res;
}


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      2SAT.hpp                                                            %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
struct SAT{
  int n,io;
  vector<int> o, cz, co;
  vector<vector<int>> d;
  SAT(int _n):n(_n), io(0){
    o.resize(2*n); d.resize(2*n);
    cz.resize(2*n); co.resize(2*n);
  }
  void dfs(int nr){
    if(cz[nr])return;
    cz[nr]=1;
    for(int ak:d[nr]) dfs(ak);
    o[io++]=nr;
  }
  bool dfs2(int nr){
    if(!cz[nr])
      return !co[nr];
    cz[nr]=0; co[nr]=1;
    for(int ak:d[nr])
      if(dfs2(ak))
        return 1;
    return 0;
  }
  vector<bool> licz(){
    R(i,2*n) if(!cz[i]) dfs(i);
    while(io--){
      if(cz[o[io]]){
        cz[o[io]] = co[o[io]] = 0;
```

```
      if(dfs2(o[io]^1)) return {};
    }
  }
  R(i,n) if(co[i*2] == co[i*2+1]) return {};
  vector<bool> res; R(i,n) res.PB(co[i*2]);
  return res;
 }
 void add(int a,bool nega,int b,bool negb){
  a *= 2; a += nega; b *= 2; b += negb;
  d[a^1].PB(b); d[b^1].PB(a);
 }
};
```

```
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Aho.hpp                                                         %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
const int ALFA = 'z' - 'a' + 1;
struct Aho{
  vector<VI> t;
  VI ds, il;
  Aho():t(1, VI(ALFA)), ds(1), il(1){}
  int new_node(){
    t.PB(VI(ALFA));
    ds.PB(0);
    il.PB(0);
    return SZ(ds) - 1;
  }
  void add(VI &a){
    int ak = 0;
    for(int z:a){
      if(t[ak][z] == 0){
        t[ak][z] = new_node();
      }
      ak = t[ak][z];
    }
    il[ak]++;
  }
  void add(string &z){
    vector<int> pom;
    for(char el:z) pom.PB(el - 'a');
    add(pom);
  }
  void aho(){
    vector<int> todo{0};
    R(i,SZ(todo)){
      int v = todo[i];
      il[v] += il[ds[v]];
      R(a,ALFA){
        if(t[v][a]){
          ds[t[v][a]] = v ? t[ds[v]][a] : 0;
          todo.PB(t[v][a]);
        }else
          t[v][a] = t[ds[v]][a];
      }
    }
  }
  int licz(VI &a){
    int res = 0, ak = 0;
    for(int el : a){
      ak = t[ak][el];
      res += il[ak];
    }
    return res;
  }
  int licz(string& z){
    VI pom;
    for(char el:z)pom.PB(el - 'a');
    return licz(pom);
  }
};
```

University of Warsaw Page 8

```
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      CentDecomp.h                                                    %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
typedef vector<VI> VVI;
// Library version handles changing weight of vertex and reading sum of weights of
// vertices not further than d from v. Adjusting it to another version should boil down
// to adjusting operations on trees / Change and Query functions. Another possible
// change is to make edges weighted, in that case trees need to be dynamic or another
// approach must be found.
struct CentDecomp {
  struct Tree {
    int M;
    vector<int> node;
    Tree(int n = 1) {
      M = 1;
      while (M <= n + 2) { M *= 2; }
      node.resize(2 * M + 5);
    }
    int Read(int a) {
      a = min(a, M - 1) + M;
      int res = node[a];
      while (a) {
        if (a % 2 == 1) { res += node[a - 1]; }
        a /= 2;
      }
      return res;
    }
    void Upd(int a, int x) {
      a += M;
      while (a) { node[a] += x; a /= 2; }
    }
  };
  /* Start of general part */
  VVI slo;
  int n;
  VI sz, dis;
  struct Info { int cent, dist, subtr; };
  vector<vector<Info>> cents;
  vector<Tree> summed;
  vector<vector<Tree>> branch;
  vector<int> wei;
  CentDecomp(VVI& slo_, int n_) {
    slo = slo_;
    n = n_;
    summed.resize(n + 2); branch.resize(n + 2); sz.resize(n + 2);
    dis.resize(n + 2); cents.resize(n + 2); wei.resize(n + 2);
    Rec(1, n);
  }
  void Rec(int v, int size) {
    v = CntSzCent(v, size, -1, 0, 0);
    dis[v] = 0;
    CntSzCent(v, size, -1, v, SZ(slo[v]));
    for (int i = 0; i < SZ(slo[v]); i++) {
      int nei = slo[v][i];
      branch[v].PB(Tree(sz[nei] + 1));
      DelEdge(nei, v);
      Rec(nei, sz[nei]);
    }
    branch[v].PB(Tree(2));
    summed[v] = Tree(sz[v]);
  }
  int CntSzCent(int v, int size, int par, int root, int subtr) {
    sz[v] = 1;
    if (root) { cents[v].PB({root, dis[v], subtr}); }
    int ret = 0;
    for (int i = 0; i < SZ(slo[v]); i++) {
      int nei = slo[v][i];
      if (v == root) { subtr = i; }
      if (par == nei) { continue; }
```

```
      dis[nei] = dis[v] + 1;
      int cnt_rec = CntSzCent(nei, size, v, root, subtr);
      sz[v] += sz[nei];
      if (cnt_rec) { ret = cnt_rec; }
    }
    if (ret == 0) {
      if (sz[v] >= size / 2) { return v; } else { return 0; }
    } else { return ret; }
  }
  void DelEdge(int v, int nei) {
    for (int i = 0; i < SZ(slo[v]); i++) {
      if (slo[v][i] == nei) {
        swap(slo[v][i], slo[v].back());
        slo[v].pop_back();
        return;
      }
    }
  }
  /* End of general part */

  /* Start of less general part */
  void Change(int v, int x) {
    int dif = x - wei[v];
    wei[v] = x;
    for (auto tr : cents[v]) {
      branch[tr.cent][tr.subtr].Upd(tr.dist, dif);
      summed[tr.cent].Upd(tr.dist, dif);
    }
  }
  int Query(int v, int d) {
    int res = wei[v];
    for (auto tr : cents[v]) {
      int rem_dis = d - tr.dist;
      if (rem_dis < 0) { continue; }
      res += summed[tr.cent].Read(rem_dis);
      res -= branch[tr.cent][tr.subtr].Read(rem_dis);
    }
    return res;
  }
  /* End of less general part */
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      CRT.hpp                                                              %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
// Chinese Remainder Theorem
#include "Headery.hpp"

LL gcd(LL a, LL b, LL& x, LL& y){
  if (a < b) return gcd(b, a, y, x);
  if (b == 0) { x = 1; y = 0; return a; }
  LL xp; LL pom = gcd(b, a % b, xp, x);
  y = xp - x * (a / b);
  return pom;
}
// Works for POSITIVE integers up to 1e18
LL mul_bin(LL a, LL b, LL P) {
  LL res = 0;
  while (b) {
    if (b % 2 == 1) { res = (res + a) % P; }
    a = (a + a) % P;
    b /= 2;
  }
  return res;
}
inline LL Adjust(LL a, LL mod) { return (a % mod + mod) % mod; }
LL INF = 1e18;
// Works for POSITIVE integers up to 1e18
pair<LL, LL> nww(LL a, LL b, LL r1, LL r2) {
  r1 = Adjust(r1, a);
  r2 = Adjust(r2, b);
```

University of Warsaw Page 9

```
  LL x, y; LL d = gcd(a, b, x, y);
  x = Adjust(x, b);
  if (r1 % d != r2 % d) { return {-1, -1}; }
  if (a / d > INF / b + 2) { return {-1, -1}; }
  LL N = a / d * b;
  LL s = mul_bin(Adjust(r2 - r1, b) / d, x, b);
  LL new_r = (r1 + mul_bin(a, s, N)) % N;
  assert(new_r % a == r1 && new_r % b == r2);
  return {N, new_r};
}
// Elements of vec are equations of form x = p.second (mod p.first)
// If solution exists than this returns a pair (N, r) such that
//      x = r (mod N); (-1, -1) otherwise
// Works for POSITIVE integers up to 1e18
pair<LL, LL> CRT(vector<pair<LL, LL>>& vec) {
  LL N = 1, r = 0;
  for (auto& p : vec) {
    assert(p.st > 0 && p.nd >= 0);
    pair<LL, LL> new_pair = nww(N, p.first, r, p.second);
    if (new_pair.first == -1) { return {-1LL, -1LL}; }
    N = new_pair.first; r = new_pair.second;
  }
  return {N, r};
}

int32_t main() {
  vector<pair<LL, LL>> vec{{8, 2}, {144,66}, {125,41}, {143,35}};
  pair<LL, LL> crt = CRT(vec);
  assert(crt == MP(2574000LL, 345666LL));
  vec = {{999999197, 174710863}, {1000000007, 174074123}};
  assert(CRT(vec) == MP(999999203999994379LL, 987654321087654321LL));
  vec = {{4, 1}, {6, 2}};
  assert(CRT(vec) == MP(-1LL, -1LL));
}


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Date.hpp                                                             %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
int dateToDay(int y, int m, int d) {   // m = [1, 12], d >= 1
  m = (m + 9) % 12;
  y -= m / 10;
  return 365 * y + y/4 - y/100 + y/400 + (m * 306 + 5) / 10 + (d - 1);
}

int weekDay(int day) { return (day + 2) % 7 + 1; }  // 1 <= result <= 7

void dayToDate(int day, int &y, int &m, int &d) {
  y = (10000 * day + 14780) / 3652425;
  d = day - (365 * y + y/4 - y/100 + y/400);
  if (d < 0) {
    y--;
    d = day - (365 * y + y/4 - y/100 + y/400);
  }
  int b = (100 * d + 52) / 3060;
  m = (b + 2) % 12 + 1;
  y += (b + 2) / 12;
  d -= (b * 306 + 5) / 10 - 1;
}


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Dominators.hpp                                                       %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"

struct Dominators{
  int n_orig, n;
  VI parent, semi, vertex, dom, ancestor, label;
  vector<VI> succ, pred, bucket;
```

```
  Dominators(int _n):n_orig(_n), n(2 * (_n + 1)), parent(n), semi(n), vertex(n), dom(n)
, ancestor(n), label(n), succ(n), pred(n), bucket(n) {
    n = n_orig;
  }
  void add_edge(int a,int b){
    a++; b++;
    succ[a].PB(b);
  }
  void COMPRESS(int v) {
    if (ancestor[ancestor[v]] != 0) {
      COMPRESS(ancestor[v]);
      if (semi[label[ancestor[v]]] < semi[label[v]]) {
        label[v] = label[ancestor[v]];
      }
      ancestor[v]=ancestor[ancestor[v]];
    }
  }
  void LINK(int v, int w) {
    ancestor[w]=v;
  }
  int EVAL(int v) {
    if(ancestor[v] == 0)
      return v;
    else {
      COMPRESS(v);
      return label[v];
    }
  }
  void DFS(int v) {
    semi[v] = ++n;
    vertex[n] = v;
    for(auto ng : succ[v]) {
      if(semi[ng] == 0) {
        parent[ng]=v;
        DFS(ng);
      }
      pred[ng].push_back(v);
    }
  }
  //dominatory z wierzcholka 0
  //zwraca vector dominatorow (-1 dla 0)
  vector<int> doit() {
    iota(ALL(label), 0);
    DFS(1);
    for (int i = n; i >= 2; --i) {
      int w = vertex[i];
      for (auto ng : pred[w]) {
        int u = EVAL(ng);
        if (semi[u] < semi[w]) { semi[w] = semi[u]; }
      }
      bucket[vertex[semi[w]]].push_back(w);
      LINK(parent[w],w);
      while (!bucket[parent[w]].empty()) {
        int v = bucket[parent[w]].back();
        bucket[parent[w]].pop_back();
        int u = EVAL(v);
        if (semi[u] < semi[v]) {
          dom[v] = u;
        } else {
          dom[v] = parent[w];
        }
      }
    }
    for (int i = 2; i <= n; ++i) {
      int w = vertex[i];
      if (dom[w] != vertex[semi[w]]) { dom[w] = dom[dom[w]]; }
    }
    dom[1] = 0;
    vector<int> res(n_orig);
    R(i,n_orig)res[i] = dom[i + 1] - 1;
    return res;
  }
```

University of Warsaw Page 10

```
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      DeBruijn.hpp                                                          %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
//Generates De Bruijn sequence containing all words from [0, alph - 1]^len
//This is version that needs to be "cycled"<=>of length alph^len not alph^len +len-1
VI de_bruijn(int len, int alph){
  VI res, lyn{0};
  while (lyn[0] != alph - 1){
    int l = SZ(lyn);
    if (len % l == 0) { R(i,l) { res.PB(lyn[i]); } }
    FOR(i, l, len - 1) { lyn.PB(lyn[i - l]); }
    while (lyn.back() == alph - 1) { lyn.pop_back(); }
    lyn.back()++;
  }
  res.PB(alph - 1);
  return res;
}                               // 1 1 -> 0
int32_t main(){                 // 4 2 -> 0000100110101111
  int len, alph;                // 3 3 -> 000100201101202102211121222
  cin>>len>>alph;
  auto dbr = de_bruijn(len, alph);
  for (int a : dbr) { cout<<a; }
  cout<<endl;
}


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Dinic.hpp                                                             %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"

struct Dinic {
  struct Edge { int v, c, inv; };
public:
  Dinic() : n(-1) {}
  void AddEdge(int a, int b, int cap, int bi_dir) {
    if (n < max(a, b)) {
      n = max(n, max(a, b));
      ResizeVectors();
    }
    e_orig[a].PB(Edge{b, cap, SZ(e_orig[b])});
    e_orig[b].PB(Edge{a, bi_dir * cap, SZ(e_orig[a]) - 1});
  }
  int MaxFlow(int s, int t) {
    if (t > n || s > n) {
      n = max(s, t);
      ResizeVectors();
    }
    e = e_orig; int result = 0;
    while (Bfs(s, t)) {
      fill_n(beg.begin(), n + 1, 0);
      result += Dfs(s, t, kInf);
    }
    return result;
  }
  vector<bool> MinCut(int s, int t) {
    assert(!Bfs(s, t));
    vector<bool> res(n + 1);
    FOR (i, 0, n) { res[i] = (dis[i] <= n); }
    return res;
  }

  vector<PII> EdgeCut(int s, int t) {
    vector<bool> left_part = MinCut(s, t);
    vector<PII> cut;
    FOR (v, 0, n) {
      for (auto edge : e_orig[v]) {
        if (edge.c != 0 && left_part[v] && !left_part[edge.v]) {
```

```
          cut.PB({v, edge.v});
        }
      }
    }
    return cut;
  }
private:
  int n;
  vector<vector<Edge>> e_orig, e;
  VI dis, beg;

  bool Bfs(int s, int t) {
    fill_n(dis.begin(), n + 1, n + 1);
    dis[s] = 0;
    VI que;
    que.push_back(s);
    REP (i, SZ(que)) {
      int v = que[i];
      for (auto edge : e[v]) {
        int nei = edge.v;
        if (edge.c && dis[nei] > dis[v] + 1) {
          dis[nei] = dis[v] + 1;
          que.push_back(nei);
          if (nei == t) { return true; }
        }
      }
    }
    return false;
  }
  int Dfs(int v, int t, int min_cap) {
    int result = 0;
    if (v == t || min_cap == 0) { return min_cap; }
    for (int& i = beg[v]; i < SZ(e[v]); i++) {
      int nei = e[v][i].v, c = e[v][i].c;
      if (dis[nei] == dis[v] + 1 && c > 0) {
        int flow_here = Dfs(nei, t, min(min_cap, c));
        result += flow_here;
        min_cap -= flow_here;
        e[v][i].c -= flow_here;
        e[nei][e[v][i].inv].c += flow_here;
      }
      if (min_cap == 0) { break; }
    }
    return result;
  }

  void ResizeVectors() {
    e_orig.resize(n + 2);
    beg.resize(n + 2);
    dis.resize(n + 2);
  }

  #ifndef int
  #warning
  #endif
  static const int kInf = 1e18; // UWAZAC, JESLI NIE INT = LONG LONG
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      DMST.hpp                                                     %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"

// Wierzcholki numerowane 0 .. N - 1; szuka drzewa ze sciezkami skierowanymi z roota
// Zwraca wektor N-elementowy prev[N]; prev[root] = -1, prev[v] = numer optymalnej
// krawedzi wchodzacej do v. getValue(compute(root)) oblicza koszt DMST.
struct DMST {
  int N;
  VI eFrom, eTo, eCost, ePrev, visited, cycle, parent;
  vector<VI> cycles, adj, curEdge;
  int Root, fstEdge;
```

University of Warsaw Page 11

```
  DMST(int V) : N(V), visited(2*V), parent(2*V), cycles(2*V), adj(2*V),
        curEdge(2*V, VI(2*V, -1)) {}

  void addEdge(int u, int v, int c, int prev = -1) {
    if (prev != -1) {
      if (curEdge[u][v] != -1) {
        int id = curEdge[u][v];
        if (eCost[id] > c) { eCost[id] = c; ePrev[id] = prev; }
        return;
      }
    }
    int id = SZ(eFrom);
    if (u == v) {
      u = v = c = -1;
    } else {
      adj[u].PB(id);
      curEdge[u][v] = id;
    }
    eFrom.PB(u); eTo.PB(v); eCost.PB(c); ePrev.PB(prev);
  }

  bool dfsCyc(int v) {
    if (v == Root) { return false; }
    visited[v] = 1;
    cycle.PB(parent[v]);
    int p = eFrom[parent[v]];
    if (visited[p] == 1) { fstEdge = parent[p]; }
    bool res = visited[p] == 1 || (!visited[p] && dfsCyc(p));
    visited[v] = 2;
    return res;
  }

  VI compute(int root) {
    Root = root;
    vector<bool> current(2 * N), onCycle(2 * N);
    VI best(2 * N);
    fill_n(current.begin(), N, true);
    int curSz = N;

    while (true) {
      fill(ALL(best), Infty);
      fill(ALL(onCycle), false);

      REP (i, 2 * N) {
        if (!current[i]) { continue; }
        for (int e : adj[i]) {
          int v = eTo[e], c = eCost[e];
          if (v != root && current[v] && c < best[v]) {
            best[v] = c; parent[v] = e;
          }
        }
      }
      fill(ALL(visited), 0);
      REP (i, 2 * N) {
        if (current[i] && !visited[i]) {
          cycle.clear();
          if (dfsCyc(i)) { break; } else { cycle.clear(); }
        }
      }
      if (!SZ(cycle)) { break; }
      cycle.erase(cycle.begin(), find(ALL(cycle), fstEdge));
      cycles[curSz] = cycle;
      for (int v : cycle) { onCycle[eFrom[v]] = true; }

      REP (v, 2 * N) {
        if (!current[v]) { continue; }
        VI edges = adj[v];
        for (int e : edges) {
          int s = eTo[e], c = eCost[e];
          if (!current[s]) { continue; }
          if (!(onCycle[v] ^ onCycle[s])) { continue; }
```

```
        if (onCycle[s]) { c -= best[s]; }
        addEdge(onCycle[v] ? curSz : v, onCycle[s] ? curSz : s, c, e);
      }
    }

    for (int v : cycle) { current[eFrom[v]] = false; }
    current[curSz++] = true;
  }

  for (int cyc = curSz - 1; cyc >= N; cyc--) {
    for (int v : cycles[cyc]) { parent[eTo[v]] = v; }
    int e = ePrev[parent[cyc]];
    parent[eTo[e]] = e;
    REP (v, 2 * N) {
      if (v != root && eFrom[parent[v]] == cyc) {
        parent[v] = ePrev[parent[v]];
      }
    }
  }
  parent[root] = -1;
  return VI(parent.begin(), parent.begin() + N);
}

int getValue(VI sol) {
  int total = 0;
  for (int i = 0; i < N; i++) { if (i != Root) { total += eCost[sol[i]]; } }
  return total;
}

const int Infty = 1e9;
};
```

```
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Duval.hpp                                                       %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
const int MAX = 200100;
string duval(string z){
  int n = SZ(z);
  z += z;
  int beg = 0;
  int len = 1;
  for(int i=1,j=1; 1; i++,j++){
    if(j == len) j = 0; // j == (i - beg) % len
    if(z[i] != z[beg + j]){
      if(z[i] > z[beg + j]){
        j = i - beg;
        len = j + 1;
      }else{
        i = beg = i - j;
        j = 0;
        len = 1;
      }
    }
    if(i - beg == n && j == 0){
      return z.substr(beg,n);
    }
  }
}
```

```
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Euler.hpp                                                       %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
// Finding Euler's cycle
// Multiedges and selfloops are no problem
#include "Headery.hpp"

struct Euler {
  struct Edge { int nei, nr; };
  vector<vector<Edge>> slo;
```

University of Warsaw Page 12

```
  VI ans, used, deg, beg;
  int e_num, n;
  Euler() : e_num(0), n(0) {}
  void AddEdge(int a, int b) {
    e_num++;
    if (a > n || b > n) {
      n = max(a, b);
      slo.resize(n + 2);
      deg.resize(n + 2);
      beg.resize(n + 2);
    }
    used.PB(0);
    slo[a].PB({b, e_num});
    slo[b].PB({a, e_num});
    deg[a]++; deg[b]++;
  }
  VI FindEuler() { // if used many times, please clear ans, beg, used
    used.PB(0);
    assert(SZ(used) > e_num);
    RE (i, n) { if (deg[i] % 2 == 1) { return VI(); } }
    Go(1);
    return ans;
  }
private:
  void Go(int v) {
    debug(v);
    while (beg[v] < SZ(slo[v])) {
      Edge& e = slo[v][beg[v]];
      beg[v]++;
      int nei = e.nei;
      if (used[e.nr]) { continue; }
      used[e.nr] = 1;
      Go(nei);
      ans.PB(nei);
    }
  }
};
```

```
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      FFT.hpp                                                         %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
const LD kPi = 2 * acos(0);

struct CD {
  LD re, im;
  CD operator=(LD a) { re = a; im = 0; return *this; }
  CD operator*(CD& z) { return {re * z.re - im * z.im, re * z.im + im * z.re}; }
  void operator*=(CD& z) { *this = (*this * z); }
  CD operator+(CD& z) { return {re + z.re, im + z.im}; }
  CD operator-(CD& z) { return {re - z.re, im - z.im}; }
  void operator/=(LD f) { re /= f; im /= f; }
};

int powMod(int a, int n, int p) {
  int res = 1;
  while (n) {
    if (n & 1) { res = ((LL)res * a) % p; }
    n >>= 1; a = ((LL)a * a) % p;
  }
  return res;
}

struct FFT {
private:
  CD *A, *B, *tmp, *res, *omega;
  int *perm;
  int maxh;
  // not needed if this is going to be used just once
  void Clear(int n) {
    REP (i, n) { A[i] = B[i] = res[i] = tmp[i] = 0; }
```

```
  }

  void fft(CD* from, CD* to, int depth, bool inv){
    int N = (1 << depth);
    for (int i = 0; i < N; i++) { to[perm[i] >> (maxh - depth)] = from[i]; }
    RE (m, depth) {
      int step = 1 << m;
      for (int pos = 0; pos < N; pos += step){
        int cur = 0;
        int delta = 1 << (maxh - m);
        if (!inv) { cur = 1 << maxh; delta *= -1; }
        CD *lft = to + pos, *rgt = lft + step / 2;
        REP (k, step / 2) {
          CD a = *lft, b = omega[cur] * *rgt;
          *lft++ = a + b; *rgt++ = a - b;
          cur += delta;
        }
      }
    }

    if (inv) { REP (i, N) { to[i] /= N; } }
  }

public:
  FFT(int deg) { // max degree of a polynomial given as input
    maxh = 0; int N = 1, h = -1;
    while (N <= 2 * deg) { maxh++; N *= 2; }
    deg = N + 20;
    A = new CD[deg];
    B = new CD[deg];
    res = new CD[deg];
    tmp = new CD[deg];
    omega = new CD[deg];
    perm = new int[deg];
    LD ang = 2 * kPi / N;
    REP (i, N + 1) { omega[i] = {cos(i * ang), sin(i * ang)}; }
    perm[0] = 0;
    RE (i, N - 1) {
      if ((i & (i - 1)) == 0) { h++; }
      perm[i] = perm[i ^ (1 << h)] | (1 << (maxh - h - 1));
    }
  }
  VI mul_less_exact(VI Q, VI R, int P) {
    int depth = 0, size = 1;
    int N = SZ(Q) + SZ(R) - 1;
    while (size < N) { depth++; size *= 2; }
    Clear(size);
    // start miejsca, w ktorym jak mozna mniejsza dokladnosc, to podmienic na komentarz
e
    // P,Q \in R[x], A = Q * (1+i)/2 + R * (1-i)/2 -> Re(A^2) = P*Q
    copy(ALL(Q), A); copy(ALL(R), B);
    // REP (i, SZ(Q)) { A[i] = CD{.5 * Q[i], .5 * Q[i]}; }
    // REP (i, SZ(R)) { A[i] = A[i] + CD{.5 * R[i], -.5 * R[i]}; }
    // fft(A, tmp, depth, false);
    // REP (i, size) tmp[i] *= tmp[i];
    fft(A, res, depth, false);
    fft(B, tmp, depth, false);
    REP (i, size) tmp[i] *= res[i];
    // koniec
    fft(tmp, res, depth, true);
    VI ans;
    REP (i, N) { ans.PB((long long)round(res[i].re) % P); }
    return ans;
  }

  VI Prepare(VI& v, int base, int bpow) {
    VI ans;
    for (int x : v) { ans.PB(bpow ? x / base : x % base); }
    return ans;
  }
  int Sum(VI& v, int P) { // debug/assert purposes only
    return accumulate(ALL(v), 0LL) % P;
```



```
  }
  VI mul_exact(VI Q, VI R, int P) {
    int base = 32000;
    int pows[] = {1, base, (int)1LL * base * base % P};
    VI ans(SZ(Q) + SZ(R) - 1);
    REP (q, 2) {
      VI W = Prepare(Q, base, q);
      REP (r, 2) {
        VI V = Prepare(R, base, r);
        // jezeli bedzie za wolno, to moÅ¼na policzyc tylko 4 transformaty w przod
        // bo teraz dla kazdej z 4 czesci jest liczona podwojnie (przyspieszenie * 2/3)
        VI C = mul_less_exact(W, V, P);
        REP (i, SZ(C)) { ans[i] = (ans[i] + 1LL * C[i] * pows[q + r]) % P; }
      }
    }
    debug(Sum(ans, P), 1LL * Sum(Q, P) * Sum(R, P) % P);    // DEBUG!!
    assert(Sum(ans, P) == 1LL * Sum(Q, P) * Sum(R, P) % P); // DEBUG!!
    return ans;
  }

  VI finv(VI Q, int coefs, int P) {
    Q.resize(coefs);
    VI R(coefs); R[0] = powMod(Q[0], P - 2, P); assert(Q[0]);
    function<void(int)> getInv = [&](int deg) {
      if (deg == 1) { return; }
      int mid = (deg + 1) / 2;
      getInv(mid);
      auto T=mul_less_exact(VI(Q.begin(),Q.begin()+deg),VI(R.begin(),R.begin()+mid),P);
      for (int i = 0; i < mid; i++) { T[i] = P - T[i + mid]; }
      T=mul_less_exact(VI(R.begin(),R.begin()+mid),VI(T.begin(),T.begin()+mid),P);
      for (int i = mid; i < deg; i++) { R[i] = T[i - mid]; }
    };
    getInv(coefs); return R;
  }

  // log(Q)=R: assert(Q[0] = 1), R[0] = 0, R'(x) = Q'(x) / Q(x)

  VI fsqrt(VI Q, int coefs, int P) {   // n log n, large constant, computes inverse in
I
    int Inv2 = (P + 1) / 2; Q.resize(coefs + 1);
    VI R(coefs), I(coefs); R[0]=(int)round(sqrt(Q[0])); I[0] = powMod(Q[0], P - 2, P);
    function<void(int)> getSqrt = [&](int deg) {
      if (deg == 1) { return; }
      int mid = (deg + 1) / 2;
      getSqrt(mid);
      auto T=mul_less_exact(VI(R.begin(),R.begin()+deg),VI(R.begin(),R.begin()+deg),P);
      T.PB(0);
      for (int i = 0; i < mid; i++) {
        T[i] = (LL)(Q[i + mid] - T[i + mid] + P) * Inv2 % P;
      }
      T = mul_less_exact(VI(T.begin(),T.begin()+mid), VI(R.begin(),R.begin()+mid),P);
      T = mul_less_exact(VI(T.begin(),T.begin()+mid), VI(I.begin(),I.begin()+mid),P);
      for (int i = mid; i < deg; i++) { R[i] = T[i - mid]; }

      T = mul_less_exact(VI(Q.begin(),Q.begin()+deg), VI(I.begin(),I.begin()+mid),P);
      for (int i = 0; i < mid; i++) { T[i] = P - T[i + mid]; }
      T = mul_less_exact(VI(I.begin(),I.begin()+mid), VI(T.begin(),T.begin()+mid),P);
      for (int i = mid; i < deg; i++) { I[i] = T[i - mid]; }
    };
    getSqrt(coefs);
    return R;
  }

  VI fexp(VI Q, int coefs, int P) {   // n log^2 n
    int sz = coefs; while (!sz || (sz & (sz - 1))) { sz++; }
    Q.resize(sz); assert(!Q[0]); VI R(sz);
    function<void(int, int)> getExp = [&](int minDeg, int maxDeg) {
      if (minDeg == maxDeg - 1) {
        R[minDeg] = minDeg ? (LL)powMod(minDeg, P - 2, P) * R[minDeg] % P : 1;
        return;
      }
      int mid = (minDeg + maxDeg) / 2;
```

```
      getExp(minDeg, mid);
      auto T = mul_less_exact(VI(R.begin() + minDeg, R.begin() + mid),
                              VI(Q.begin(), Q.begin() + (maxDeg - minDeg)), P);
      for (int i = mid; i < maxDeg; i++) { R[i] = (T[i - minDeg] + R[i]) % P; }
      getExp(mid, maxDeg);
    };
    for (int i = 0; i < sz; i++) { Q[i] = ((LL)Q[i] * i) % P; }
    getExp(0, sz);
    return VI(R.begin(), R.begin() + coefs);
  }
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Gauss.hpp                                                           %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
int powMod(int a, int n, int Mod) {
  int res = 1; while (n) {
    if (n & 1) { res = ((LL)res * a) % Mod; }
    a = ((LL)a * a) % Mod; n >>= 1;
  } return res;
}
int invMod(int a, int Mod) { return powMod(a, Mod - 2, Mod); }

/* Zwraca {det, rank}. */
PII gaussMod(vector<VI> matrix, int Mod) {
  int N = SZ(matrix), M = SZ(matrix[0]);
  vector<bool> visited(N);
  VI wrow(M);
  int rank = M, det = 1, ssign = 0;

  REP (col, M) {
    int which = -1;
    REP (row, N) {
      if (visited[row] || !matrix[row][col]) { continue; }
      which = row; break;
    }
    if (which == -1) { rank--; continue; }
    wrow[col] = which;
    visited[which] = true;
    det = ((LL)det * matrix[which][col]) % Mod;
    REP (row, N) {
      if (row == which || !matrix[row][col]) { continue; }
      int coef = Mod - ((LL)matrix[row][col] * invMod(matrix[which][col], Mod)) % Mod;
      REP (c, M) {
        matrix[row][c] = (matrix[row][c] + (LL)matrix[which][c] * coef) % Mod;
      }
    }
  }
  if (rank != M || N != M) { return {0, rank}; }
  REP (i, N) { FOR (j, i, N - 1) { if (wrow[i] > wrow[j]) { ssign++; } } }
  if (ssign & 1) { det = Mod - det; }
  return {det, N};
}

/* Bierze macierz N x M z N rownaniami i M zmiennymi, wektor na N wartosci, zwraca
 * jakikolwiek wektor M rozwiazan (lub wektor pusty, gdy uklad sprzeczny) */
const LD kEps = 1E-10;
vector<LD> solveSystem(vector<vector<LD>> coefs, vector<LD> values) {
  int N = SZ(coefs), M = SZ(coefs[0]);
  vector<bool> vis(N);
  VI wrow(M, -1);
  REP (col, M) {
    int which = -1; LD bAbs = kEps;
    REP (row, N) {
      if (!vis[row] && abs(coefs[row][col]) > bAbs) {
        which = row; bAbs = abs(coefs[row][col]);
      }
    }
    if (which == -1) { continue; }
    vis[which] = true;
    wrow[col] = which;

    REP (row, N) {
      if (row == which) { continue; }
      LD coef = -coefs[row][col] / coefs[which][col];
      REP (c, M) { coefs[row][c] += coef * coefs[which][c]; }
      values[row] += coef * values[which];
    }
  }
  REP (row, N) { if (!vis[row] && abs(values[row]) > kEps) { return {}; } }
  vector<LD> result(M);
  REP (col, M) {
    if (wrow[col] != -1) {
      result[col] = values[wrow[col]] / coefs[wrow[col]][col];
    }
  }
  return result;
}

const int Mod = 1e9 + 7;
struct Matrix {
  int N; vector<VI> data;
  Matrix(int size) : N(size) { data = vector<VI>(N, VI(N, 0)); }

  // redukcja kolumny 'col' (zmiennej) za pomoca wiersza 'row' (rownania)
  // (element Gaussa) wykonana na macierzy mat
  void reduce(int row, int col, Matrix& mat) const {
    assert(N == mat.N && data[row][col]);
    REP (r, N) {
      if (r == row || !data[r][col]) { continue; }
      int coef = invMod(data[row][col], Mod);
      coef = (Mod - (LL)coef * data[r][col] % Mod) % Mod;

      REP (c, N) {
        mat.data[r][c] = (mat.data[r][c] + (LL)mat.data[row][c] * coef) % Mod;
        //mat.data[r][c] = add_mod<Mod>(mat.data[r][c],
        //                            mul_mod<Mod>(mat.data[row][c], coef));
      }
    }
  }
  // zwraca pare {macierz, rzad} (jesli second<N, first=cokolwiek)
  pair<Matrix, int> invert() const {
    Matrix oper(*this), result(N), orderedResult(N);
    int rank = 0;
    REP (i, N) { result.data[i][i] = 1; }
    vector<bool> used(N);
    VI order;

    REP (col, N) {
      int reduceRow = -1;
      REP (row, N) {
        if (used[row] || !oper.data[row][col]) { continue; }
        reduceRow = row; break;
      }

      if (reduceRow == -1) { continue; }
      rank++;
      oper.reduce(reduceRow, col, result);
      oper.reduce(reduceRow, col, oper);
      order.push_back(reduceRow);
      used[reduceRow] = true;
    }

    REP (i, SZ(order)) { orderedResult.data[i] = result.data[order[i]]; }
    return {orderedResult, rank};
  }
};
```

```
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      GomoryHu.hpp                                                         %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Dinic.hpp"                  // int N, M; cin >> N >> M; GomoryHu gomory(N);
                                      // REP (i, M) { int u, v, c; cin >> u >> v >> c;
struct GomoryHu {                     //               gomory.addEdge(u, v, c); }
  vector<vector<PII>> graph, tree; // auto V = gomory.run();
  vector<VI> nodes;                   // REP (i, N) for (auto P : V[i])
  vector<bool> visited;               //   if(P.st>i){cout<<i<<" "<<P.st<<" "<<P.nd<<"\n";
}
  VI groupId, contrId;
  int wnode, n;
  GomoryHu(int N) : graph(N), visited(N), groupId(N), contrId(N), tree(N), n(N) {}

  void addEdge(int u, int v, int cap) {
    graph[u].emplace_back(v, cap);
    graph[v].emplace_back(u, cap);
  }
  void dfs(int v, int type) {
    visited[v] = true; contrId[v] = type;
    for (auto P : tree[v]) { if (!visited[P.st]) { dfs(P.st, type); } }
  }

  vector<vector<pair<int, int>>> run() {
    vector<int> allNodes(n);
    iota(ALL(allNodes), 0);
    nodes = vector<VI>{allNodes};
    tree = vector<vector<PII>>(n);
    fill(ALL(groupId), 0);

    for (int step = 1; step < n; step++) {
      Dinic dinic;
      for (int i = 0; i < SZ(nodes); i++) {
        if (SZ(nodes[i]) > 1) { wnode = i; break; }
      }
      fill(ALL(visited), false);

      visited[wnode] = true;
      for (auto P : tree[wnode]) { dfs(P.st, nodes[P.st][0]); }
      for (int v = 0; v < n; v++) {
        int a = groupId[v] == wnode ? v : contrId[groupId[v]];
        for (auto& P : graph[v]) {
          int b = groupId[P.st] == wnode ? P.st : contrId[groupId[P.st]];
          if (a != b) { dinic.AddEdge(a, b, P.nd, 0); }
        }
      }
      dinic.AddEdge(n-1, n-1, 0, 0);

      int a = nodes[wnode][0], b = nodes[wnode][1], f = dinic.MaxFlow(a, b);
      auto cut = dinic.MinCut(a, b);

      for (int v = 0; v < step; v++) {
        if (v == wnode) { continue; }
        for (auto& P : tree[v]) {
          if (P.st == wnode && !cut[contrId[v]]) { P.st = step; }
        }
      }
      vector<PII> PA, PB;
      for (auto& P : tree[wnode]) { (cut[contrId[P.st]] ? PA : PB).PB(P); }
      tree[wnode] = PA; tree[step] = PB;
      tree[wnode].emplace_back(step, f);
      tree[step].emplace_back(wnode, f);
      VI A, B;
      for (int v : nodes[wnode]) {
        (cut[v] ? A : B).push_back(v);
        if (!cut[v]) { groupId[v] = step; }
      }
      nodes[wnode] = A;
      nodes.push_back(B);
    }

    vector<vector<PII>> res(n);
```

University of Warsaw Page 15

```
    for (int i = 0; i < n; i++) {
      for (auto P : tree[i]) { res[nodes[i][0]].emplace_back(nodes[P.st][0], P.nd); }
    }
    return res;
  }
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      HLDWithRev.hpp                                                       %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
typedef vector<VI> VVI;
// This version allows to keep an order of edges/vertices on a path
struct Node { // not general at all
  int left_len, mid_val, right_len, whole;
  void Rev() { // usually no need to
    if (!whole) { swap(left_len, right_len); }
  }
  int Eval(); // only in this task
};
Node Merge(Node a, Node b);

struct HLD {
  struct Tree {
    Tree(int nn) {
      M = 1;
      while (M <= nn) {
        M *= 2;
      }
      node.resize(2 * M + 5);
    }
    Node ReadNode(int a, int b) {
      a += M; b += M;
      if (a == b) { return node[a]; }
      Node left = node[a];
      Node right = node[b];
      while (a / 2 != b / 2) {
        if (a % 2 == 0) {
          left = Merge(left, node[a + 1]);
        }
        if (b % 2 == 1) {
          right = Merge(node[b - 1], right);
        }
        a /= 2; b /= 2;
      }
      return Merge(left, right);
    }

    void Upd(int a) {
      a += M;
      node[a] = {1, 0, 0, 1}; // new value of vertex, usually need one more arg
      a /= 2;
      while (a) {
        node[a] = Merge(node[2 * a], node[2 * a + 1]);
        a /= 2;
      }
    }
    int M; vector<Node> node;
  };

public:
  /* Start of general part */
  HLD(VVI& slo_, int n_, int on_vertices_, int root_ = 1) {
    init = 0;
    n = n_;
    on_vertices = on_vertices_;
    assert(SZ(slo_) >= n_);
    slo = slo_; root = root_;
    path.resize(n + 5); wh_path.resize(n + 5); ord.resize(n + 5);
    sz.resize(n + 5); pre.resize(n + 5); par.resize(n + 5); path_pot.resize(n + 5);
    path_cnt = 0; d = 0;
```

```
    Dfs(root);
    tree.PB(Tree(1));
    for (int p = 1; p <= path_cnt; p++) { tree.PB(Tree(SZ(path[p]))); }
    init = 1; // First moment when we can do any updates
  }

  bool IsAnc(int v, int u) { return pre[v]<=pre[u] && pre[u] <= pre[v] + sz[v] - 1; }

  int n, path_cnt, d, root, on_vertices, init, last_lca;
  VVI slo, path;
  vector<Tree> tree;
  VI wh_path, ord, sz, pre, par, path_pot;

  void Dfs(int v) {
    sz[v] = 1; d++; pre[v] = d;
    int largest_son = 0;
    for (auto nei : slo[v]) {
      if (pre[nei]) { continue; }
      par[nei] = v;
      Dfs(nei);
      sz[v] += sz[nei];
      if (sz[nei] > sz[largest_son]) { largest_son = nei; }
    }
    if (largest_son == 0) {
      path_cnt++;
      path[path_cnt].PB(v);
      wh_path[v] = path_cnt;
    } else {
      wh_path[v] = wh_path[largest_son];
      ord[v] = ord[largest_son] + 1;
      path[wh_path[v]].PB(v);
    }
  }
  struct Info {
    // a - bottom of path, b - top (smaller depth)
    // they denote indices on given path (from bottom to top)
    int path, a, b;
  };
  vector<vector<Info>> GetPath(int a, int b) {
    assert(init);
    vector<Info> res[2];
    for (int tr = 0; tr < 2; tr++) {
      while (1) {
        int p = wh_path[a];
        int mi = ord[a];
        int ma = SZ(path[p]) - 1;
        if (IsAnc(path[p].back(), b)) {
          int kl = mi; int kp = ma;
          while (kl <= kp) {
            int aktc = (kl + kp) / 2;
            if (IsAnc(path[p][aktc], b)) {
              kp = aktc - 1; ma = aktc;
            } else {
              kl = aktc + 1;
            }
          }
        }
        res[tr].PB({p, mi, ma});
        last_lca = path[p][ma];
        if (IsAnc(path[p].back(), b)) { break; }
        a = par[path[p].back()];
      }

      // removing occurance of LCA
      if (tr == 1 || !on_vertices) {
        if (res[tr].back().a == res[tr].back().b) {
          res[tr].pop_back();
        } else {
          res[tr].back().b--;
        }
      }
      swap(a, b);
    }
    return {res[0], res[1]};
  }

  /* End of general part */

  //Update on path a<->b
  //Current version assumes ab is an edge (and weirdly no args for update)
  //If we need to update just one vertex v then we should be asking for GetPath(v, v)
  //Updates for whole paths then we need to call tree[tr.st].Upd(tr.a, tr.b, val)
  void Upd(int a, int b) {
    auto sub = GetPath(a, b);
    for (auto subpaths : sub) {
      for (auto tr : subpaths) {
        tree[tr.path].Upd(tr.a);
      }
    }
  }
  //Query on path a<->b
  int Query(int a, int b) {
    auto subpaths = GetPath(a, b);
    Node no[2] = {{0, 0, 0, 1}, {0, 0, 0, 1}}; // <- here put default values
    for (int wh = 0; wh < 2; wh++) {
      for (auto tr : subpaths[wh]) {
        no[wh] = Merge(no[wh], tree[tr.path].ReadNode(tr.a, tr.b));
      }
    }
    no[1].Rev(); // usually no need to
    Node R = Merge(no[0], no[1]);
    return R.Eval();
  }
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Hungarian.hpp                                                         %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "../Headery.hpp"

const int inf = 1e12;
struct Hungarian{
  int n,m;
  VI u, v, p, way;
  int run(vector<VI> t){
    //przenumerowanie
    vector<VI> a(SZ(t) + 1, vector<int>(SZ(t[0]) + 1));
    R(i, SZ(t))R(j, SZ(t[0])){
      a[i+1][j+1] = t[i][j];
    }
    //--------------
    n = SZ(a) - 1;
    m = SZ(a[0]) -1;
    u.resize(n + 1);
    v.resize(m + 1);
    p.resize(m + 1);
    way.resize(m + 1);
    for(int i = 1; i <=n; i++){
      p[0] = i;
      int j0 = 0;
      VI minv(m+1, inf);
      VI used(m+1, false);
      do{
        used[j0] = true;
        int i0 = p[j0], delta = inf, j1;
        for(int j=1; j<=m; ++j) if(!used[j]){
          int cur = a[i0][j] - u[i0] - v[j];
          if(cur < minv[j]){
            minv[j] = cur;
            way[j] = j0;
          }
          if(minv[j] < delta){
            delta = minv[j];
```

```
          j1 = j;
        }
      }
      R(j, m+1){
        if(used[j]){
          u[p[j]] += delta;
          v[j] -= delta;
        }else
          minv[j] -= delta;
      }
      j0 = j1;
    }while(p[j0] != 0);
    do{
      int j1 = way[j0];
      p[j0] = p[j1];
      j0 = j1;
    }while(j0);
  }
  return -v[0];
}

VI mathing(){
  VI res(n);
  for(int j = 1; j <= m; j++){
    if(p[j]) res[p[j] - 1] = j - 1;
  }
  return res;
}
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      KS.hpp                                                              %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"

using triple = tuple<int,int,int>;

// sortuje indeksy z 'src' do 'dest' po kluczu 'keys' z zakresu [0..K]
void ksRadixPass(const VI& src, VI& dest, int size, const VI& keys, int K){
  VI cnt(K+1, 0);
  for(int i = 0; i < size; i++) { cnt[keys[src[i]]]++; }
  int prefSum = 0;
  for(int i = 0; i <= K; i++){
    int tmp = cnt[i];
    cnt[i] = prefSum;
    prefSum += tmp;
  }
  for(int i = 0; i < size; i++){
    int& pos = cnt[keys[src[i]]];
    dest[pos] = src[i];
    pos++;
  }
}

// znajduje tablice sufiksowa src[0..n-1] z kluczy {1..K}^n do dest[]
// koniecznosc: src[n..n+2] = 0 oraz n >= 2
void ksSuffixArray(const VI& src, VI& dest, int N, int K){
  int n0 = (N + 2) / 3, n2 = N / 3,  n02 = n0 + n2;
  VI src12(n02 + 3, 0), dest12(n02 + 3, 0), src0(n0, 0), dest0(n0, 0);
  // tworzymy liste indeksow o resztach 1 i 2 mod 3
  int ptr = 0;
  for(int i = 0; i < N + (N % 3 == 1); i++) {
    if(i % 3 != 0) src12[ptr++] = i;
  }
  // sortujemy pozycyjnie reszty 1 i 2
  ksRadixPass( src12, dest12, n02, VI(src.begin()+2, src.end()), K);
  ksRadixPass(dest12,  src12, n02, VI(src.begin()+1, src.end()), K);
  ksRadixPass( src12, dest12, n02, src,                          K);
  // przyporzadkowujemy trojkom (poczynajac od indeksow) leksykograficzne numerki
  int numNames = 0;
  triple biggestTriple = triple(-1, -1, -1);
```

University of Warsaw Page 17

```
  for (int i = 0; i < n02; i++) {
    int pos = dest12[i];
    triple newTriple = triple(src[pos], src[pos+1], src[pos+2]);
    if (newTriple != biggestTriple) {
      numNames++;
      biggestTriple = newTriple;
    }
    if (pos % 3 == 1) {
      src12[pos / 3] = numNames;
    } else {
      src12[pos / 3 + n0] = numNames;
    }
  }
  // rekurencja, gdy nie ma unikatowych nazw
  if (numNames < n02) {
    ksSuffixArray(src12, dest12, n02, numNames);
    // przyporzadkowujemy kolejnym wartosciom unikatowe nazwy
    for (int i = 0; i < n02; i++) { src12[dest12[i]] = i + 1; }
  } else {
    // mozna sobie stworzyc prosciutko tablice sufiksowa
    for (int i = 0; i < n02; i++) { dest12[src12[i] - 1] = i; }
  }

  // sortujemy sufiksy podzielne przez 3 po pierwszym znaczku
  ptr = 0;
  for (int i = 0; i < n02; i++)
    if (dest12[i] < n0) src0[ptr++] = 3 * dest12[i];
  ksRadixPass(src0, dest0, n0, src, K);
  // laczymy
  for (int p = 0, t = (N % 3 == 1), k=0; k < N; k++) {
    int i = dest12[t] < n0 ? dest12[t] * 3 + 1 : (dest12[t] - n0) * 3 + 2,
        j = dest0[p];

    bool take0;
    if (dest12[t] < n0) {
      take0 = MP(src[i], src12[dest12[t] + n0]) < MP(src[j], src12[j / 3]);
    } else {
      take0 = triple(src[i], src[i + 1], src12[dest12[t] - n0 + 1]) <
              triple(src[j], src[j + 1], src12[j / 3 + n0]);
    }

    if (take0) {
      dest[k] = i;
      t++;
      if (t == n02) {
        k++;
        while (p < n0) {
          dest[k] = dest0[p]; k++; p++;
        }
      }
    } else {
      dest[k] = j; p++;
      if (p == n0) {
        k++;
        while (t < n02) {
          if (dest12[t] < n0) {
            dest[k] = dest12[t] * 3 + 1;
          } else {
            dest[k] = (dest12[t] - n0) * 3 + 2;
          }
          k++; t++;
        }
      }
    }
  }
}

VI suffixArray(VI source) {
  int N = SZ(source), K = *max_element(ALL(source));
  if (N == 1) { return {0}; }
  for (int& v : source) { v++; }
  for (int i = 0; i < 3; i++) { source.PB(0); }
```

```
  VI result(N + 3);
  ksSuffixArray(source, result, N, K + 2);
  result.resize(N);
  return result;
}
/* {1, 2, 1, 1, 2} -> {2, 3, 0, 4, 1} */


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Low.hpp                                                              %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
struct Low {
  VI low, vis1, vis2, pre, ojc, art, wejdwu, most;
  vector<VI> slo, dwu;
  int aktdwu, d; int n;
  Low(int n_) {
    n = n_;
    low.resize(n + 2); vis1.resize(n + 2); vis2.resize(n + 2); pre.resize(n + 2);
    ojc.resize(n + 2); art.resize(n + 2); wejdwu.resize(n + 2); most.resize(n + 2);
    slo.resize(n + 2); dwu.resize(n + 2);
    aktdwu = d = 0;
  }
  /* Mosty to krawedzie od v do ojc[v] takie, ¿e most[v]=1
  Punkty artykulacji to te, dla ktorych art[v]=1
  Numer dwuspojnej krawedzi v->slo[v][i] to dwu[v][i] */
  void dfs1(int v) {
    d++;
    vis1[v] = 1; pre[v] = d; low[v] = pre[v];
    REP (i, SZ(slo[v])) {
      int nei = slo[v][i];
      if (ojc[v] == nei) { continue; }
      if (vis1[nei] == 0) { // drzewowa w dol
        ojc[nei] = v;
        dfs1(nei);
        mini(low[v], low[nei]);
        if (low[nei] >= pre[v]) { art[v] = 1; }
        if (low[nei] > pre[v]) { most[nei] = 1; }
      } else { // niedrzewowa w gore
        mini(low[v], pre[nei]);
      }
    }
  }

  void dfs2(int v) { // tylko do wyliczania dwuspojnych!
    vis2[v] = 1;
    REP (i, SZ(slo[v])) {
      int nei = slo[v][i];
      if (ojc[v] == nei) { dwu[v][i] = wejdwu[v]; debug(v, nei); continue; }
      if (vis2[nei] == 0) {
        if (low[nei] >= pre[v]) {
          aktdwu++; wejdwu[nei] = aktdwu;
        } else {
          wejdwu[nei] = wejdwu[v];
        }
        dwu[v][i] = wejdwu[nei];
        dfs2(nei);
      } else {
        if (pre[v] < pre[nei]) {
          dwu[v][i] = wejdwu[nei];
        } else {
          dwu[v][i] = wejdwu[v];
        }
      }
    }
  }
  void AddEdge(int a, int b) {
    assert(a && b);
    slo[a].PB(b); slo[b].PB(a);
    dwu[a].PB(0); dwu[b].PB(0);
  }
  void LowGo() {
```



```
    int st = 1;
    d = 0; aktdwu = 0; wejdwu[st] = 0;
    dfs1(st);
    int licz = 0;
    RE (i, n) { if(ojc[i] == st) { licz++; } }
    art[st] = (licz > 1);
    dfs2(st);
  }
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Splay.hpp                                                            %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
struct node{
  node *l, *r, *p;
  int size,val,sum;
  bool flip;
  node(int _val): l(0), r(0), p(0), val(_val), size(1), sum(_val), flip(0){}
  virtual void update(){
    size = 1;                // Wszystkie funckje nalezy wykonywac na korzeniu/korzeniach
    sum = val;               // drzewa Splay. Zwracaja one nowy korzen/korzenie.
    if(l){                   // Jesli skadinad mamy wskaznik do wierzcholka, nalezy
      size += l->size;       // wykonac na nim splay(). Remove() nie zwalnia pamieci.
      sum += l->sum;         // Usuwany wierzcholek staje sie korzeniem
    }                        // jednoelementowego drzewa.
    if(r){
      size += r->size;
      sum += r->sum;
    }
  }
  void touch(){
    if(flip){
      swap(l, r);
      if(l) l->flip ^= 1;
      if(r) r->flip ^= 1;
      flip = 0;
    }
  }
  void touch_path(){
    if(p) p->touch_path();
    touch();
  }
  node*& get_child(bool right){
    return right ? r : l;
  }
  static void add_child(node* x, node* q, bool right){
    if(x) x->get_child(right) = q;
    if(q) q->p = x;
  }
  inline bool is_right(){
    return p && p->r == this;
  }
  void rotate(){
    if(!p) return;
    node *oldp = p;
    bool right = is_right();
    add_child(p->p, this, p->is_right());
    add_child(oldp, get_child(!right), right);
    add_child(this, oldp, !right);
    oldp->update();
    update();
  }
  void splay_(){
    while(p){
      if(is_right() ^ p->is_right())
        rotate();
      else
        p->rotate();
      rotate();
    }
```

```
  }
  void splay(){ //dla nieodwracalych splay'ow zastapic przez splay_
    touch_path();
    splay_();
  }
  void set_val(int nval){
    val = nval;
    update();
  }
  void reverse(){
    flip = !flip;
  }
  node* get_first(){
    node* res = this;
    while(1){
      res->touch();
      if(!res->l) break;
      res = res->l;
    }
    res->splay_();
    return res;
  }
  node* remove() {
    if(l) l->p = nullptr;
    if(r) r->p = nullptr;
    node* root = join(l, r);
    l = r = nullptr;
    return root;
  }
  static node* join(node* a, node* b){
    if(!a) return b;
    while(1){
      a->touch();
      if(!a->r) break;
      a = a->r;
    }
    a->splay_();
    add_child(a, b, true);
    a->update();
    return a;
  }
  node* get_kth(int k){
    assert(size > k);
    node* res = this;
    while(1){
      res->touch();
      if(res->l){
        if(res->l->size > k){
          res = res->l;
          continue;
        }else
          k -= res->l->size;
      }
      if(k == 0){
        res->splay_();
        return res;
      }
      k--;
      res = res->r;
    }
  }
  pair<node*, node*> split(int k){
    if(k == 0) return {nullptr, this};
    if(k >= size) return {this, nullptr};
    node* x = get_kth(k);
    node* res = x->l;
    x->l->p = nullptr;
    x->l = nullptr;
    x->update();
    return {res, x};
  }
  ~node(){ delete l; delete r; }
```

University of Warsaw Page 19

```
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      LinkCut.hpp                                                          %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Splay.hpp"
// dalszy element splay'a jest nizej w drzewie
struct LCnode : node{
  int st_size, base_size;
  LCnode(int val,bool ver):node(val), st_size(ver), base_size(ver){};
  virtual void update(){
    node::update();
    st_size = base_size;
    if(l) st_size += ((LCnode*) l)->st_size;
    if(r) st_size += ((LCnode*) r)->st_size;
  }
  void LCsplay(){
    node* ak = this;
    node* par = ak->p;
    while(par && (par->l == ak || par->r == ak)){
      ak = par;
      par = ak->p;
    }
    ak->p = nullptr;
    splay();
    p = par;
  }
  void access(){
    node* right = nullptr;
    LCnode* cur = this;
    while(cur){
      cur->LCsplay();
      //--- zapytania o poddrzewo
      if(cur->r) cur->base_size += ((LCnode*) cur->r)->st_size;
      if(right) cur->base_size -= ((LCnode*) right)->st_size;
      //---
      cur->r = right;
      cur->update();
      right = cur;
      cur = (LCnode*)cur->p;
    }
    splay();
  }
  void to_root(){
    access();
    reverse();
    touch();
  }
  void link(LCnode* par){
    to_root();
    p = par;
    //--- zapytania o poddrzewo
    par->to_root();
    par->base_size += st_size;
    par->update();
    //---
  }
  void get_path(LCnode* v){
    v->to_root();
    access();
  }
  void cut(LCnode* v){
    get_path(v);
    v->p = l = nullptr;
    update();
  }
  bool connected(LCnode* v){
    get_path(v);
    return get_first() == v;
  }
  ~LCnode(){
```

```
    l = r = nullptr;
  };
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Manacher.hpp                                                   %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"

struct Manacher {
  VI par, npar;
  Manacher(string in) {       // IGNORES FIRST LETTER - in[0] must be space
    assert(in[0] == ' ');     // par[i] = k <=> [i - k + 1, i + k] is maximal palindrom
e
    int orig_n = SZ(in) - 1;  // npar[i] = k <=> [i - k, i + k] is maximal palindrome
    string s = " #";
    for (int i = 1; i <= orig_n; i++) {
      s += in[i]; s += '#';
    }
    s += '$';
    int new_n = SZ(s) - 2;
    npar.resize(new_n + 2);
    int furth_beg = 0; int furth_end = 0;
    for (int i = 1; i <= new_n; i++) {
      if (furth_end < i) { furth_beg = i; furth_end = i; }
      int corr_npar = furth_beg + furth_end - i;
      if (furth_end > i + npar[corr_npar]) {
        npar[i] = npar[corr_npar];
      } else {
        npar[i] = furth_end - i;
        furth_beg = i - npar[i];
        while (s[furth_beg - 1] == s[furth_end + 1]) {
          furth_beg--; furth_end++; npar[i]++;
        }
      }
    }
    par.resize(orig_n + 2);
    for (int i = 1; i <= orig_n; i++) {
      if (i < orig_n) {
        par[i] = npar[2 * i + 1] / 2;
      }
      npar[i] = npar[2 * i] / 2;
    }
    npar.resize(orig_n + 2);
  }
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      MaxFlowMinCost.hpp                                             %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"

class MinCostFlow {
  struct MCEdge { int to, cost, flow; MCEdge* next; };
  const int Infty = 1e9 + 100;
  vector<vector<MCEdge*>> adjList;
  int N, Source, Sink;
  VI dist;
  vector<MCEdge*> prev, Edges;

  void spfa() {
    queue<int> Q; vector<bool> onQueue(N); fill(ALL(dist), Infty);
    Q.push(Source); onQueue[Source] = true; dist[Source] = 0;
    while (!Q.empty()) {
      int v = Q.front(); Q.pop(); onQueue[v] = false;
      for (MCEdge *E : adjList[v]) {
        int s = E->to;
        if (E->flow == 0) { continue; }

        int newDist = dist[v] + E->cost;
```



```
        if (newDist < dist[s]) {
          dist[s] = newDist;
          prev[s] = E->next;
          if (!onQueue[s]) {
            Q.push(s); onQueue[s] = true;
          }
        }
      }
    }
  }

  int reduce_cost() {
    REP (v, N) { for (MCEdge* E: adjList[v]){ E->cost += dist[v] - dist[E->to]; } }
    return dist[Sink];
  }
  void dijkstra_shortest_path(){
    fill(ALL(dist), Infty); dist[Source] = 0;
    priority_queue<PII> Q; Q.push(make_pair(0, Source));

    while (!Q.empty()){
      int dst = -Q.top().st, v = Q.top().nd; Q.pop();
      if (dst != dist[v]) { continue; }
      for (MCEdge* E: adjList[v]){
        int s = E->to;
        if (!E->flow) { continue; }
        int newDist = dist[v] + E->cost;
        if(newDist < dist[s]){
          dist[s] = newDist;
          prev[s] = E->next;
          Q.push({-newDist, s});
        }
      }
    }
  }
public:
  MinCostFlow(){}
  MinCostFlow(int _N) : N(_N), dist(_N), prev(_N), adjList(_N) {}
  ~MinCostFlow(){ for (MCEdge* E: Edges) { delete E; } }

  void fit(int size) {
    if (size > N) {
      N = size;
      dist.resize(size); prev.resize(size); adjList.resize(size);
    }
  }

  void add_edge(int u, int v, int flow, int cost){
    fit(max(u, v) + 1);
    MCEdge *E1 = new MCEdge{v, cost, flow}, *E2 = new MCEdge{u, -cost, 0};
    Edges.PB(E1); Edges.PB(E2);
    E1->next = E2; E2->next = E1;
    adjList[u].PB(E1); adjList[v].PB(E2);
  }

  PII get_min_cost_flow(int s, int t){
    fit(max(s, t) + 1);
    Source = s; Sink = t;
    int cost = 0, flow = 0;
    spfa();
    if(dist[Sink] > Infty / 2) { return {0,0}; }
    int sinkCost = reduce_cost();

    while (true) {
      dijkstra_shortest_path();
      if(dist[Sink] > Infty / 2) { break; }
      sinkCost += reduce_cost();

      int maxSend = Infty;
      for(int v = Sink; v != Source; v = prev[v]->to){
        maxSend = min(maxSend, prev[v]->next->flow);
      }
      for(int v = Sink; v != Source; v = prev[v]->to){
```

```
        MCEdge *E1 = prev[v], *E2 = E1->next;
        E1->flow += maxSend; E2->flow -= maxSend;
      }
      flow += maxSend;
      cost += maxSend * sinkCost;
    }
    return {flow, cost};
  }
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      NearestPoints.hpp                                              %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
#define VAR(v,i) __typeof(i) v=(i)
#define FOREACH(i,c) for(VAR(i,(c).begin());i!=(c).end();++i)
const int INF = 1e9 + 1;
#define POINTT int // Dla wspolrzednych punktu (int lub double)
#define POINTR LL // Dla wynikow operacji - pole, iloczyn wektorowy (LL lub double)
struct POINT {
  POINTT x, y;
  bool operator==(POINT& a) {return a.x==x && a.y==y;}
};
bool POINTxSort(POINT *a, POINT *b) {return a->x == b->x ? a->y < b->y : a->x < b->x;}
bool POINTySort(POINT *a, POINT *b) {return a->y == b->y ? a->x < b->x : a->y < b->y;}
struct NearestPoints {
  vector<POINT*> L;
  POINT *p1, *p2; // Points that are the nearest to each other
  POINTR dist;    // Square of the distance between p1 and p2
  POINTR sqr(POINTT a) {return (POINTR)(a)*(POINTR)(a);}
  void Upd(POINT* x, POINT* y) {
    POINTR k;
    if (dist > (k = sqr(x->x - y->x) + sqr(x->y - y->y))) {
      dist = k; p1 = x; p2 = y;
    }
  }
  void Filter(vector<POINT*> &V, POINTR p) {
    int s=0;
    REP(x, V.size()) { if (sqr(V[x]->x-p) <= dist) { V[s++] = V[x]; } }
    V.resize(s);
  }
  void Calc(int p, int k, vector<POINT*> &ys) {
    if (k-p > 1) {
      vector<POINT*> lp, rp;
      int c = (k+p-1)/2;
      FOREACH(it, ys) { if (POINTxSort(L[c], *it)) rp.PB(*it); else lp.PB(*it); }
      Calc(p, c+1, lp); Calc(c+1, k, rp);
      Filter(lp, L[c]->x); Filter(rp, L[c]->x);
      int px = 0;
      FOREACH(it, lp) {
        while (px < SZ(rp)-1 && rp[px+1]->y < (*it)->y) px++;
        FOR(x, max(0LL, px-2), min((int)rp.size()-1, px+2)) Upd(*it, rp[x]);
      }
    }
  }
  NearestPoints(vector<POINT> &p) {
    FOREACH(it, p) L.PB(&(*it));
    sort(ALL(L), POINTxSort);
    FOR(x,1,SZ(L)-1)
      if (L[x-1]->x == L[x]->x && L[x-1]->y == L[x]->y) {
        dist=0; p1=L[x-1]; p2=L[x]; return;
      }
    dist = (POINTR)(INF)*(POINTR)(INF);
    vector<POINT*> v = L;
    sort(ALL(v), POINTySort);
    Calc(0, L.size(), v);
  }
};
/*int32_t main() {
  srand(21798412);   // (381, 497) (389, 504). Dist^2 = 113
  vector<POINT> v; int s=100;
```

University of Warsaw Page 21

```
  v.resize(s);
  REP(y,s) { v[y] = POINT{rand() % 1000, rand() % 1000}; }
  NearestPoints a(v);
  printf("(%lld, %lld) (%lld, %lld). Dist^2 = %lld\n", a.p1->x, a.p1->y, a.p2->x,
      a.p2->y, a.dist);
}*/


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Numbers.hpp                                                    %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"

template<class T, class U, class Op> T fastop(T a, U b, T r, const Op &op) {
  while (b > 0) {
    if (b % 2 == 1) { r = op(r, a); }
    a = op(a, a);
    b /= 2;
  }
  return r;
}

// MILLER-RABIN
template<class T> bool witness(T wit, T n) {
  if (wit >= n) { return false; }
  auto addmod = [&](T a, T b) { return (a + b) % n; };
  auto mulmod = [&](T a, T b) { return fastop(a, b, T(0), addmod); };
  auto powmod = [&](T a, T b) { return fastop(a, b, T(1), mulmod); };
  int s; T t;
  for (s = 0, t = n - 1; t % 2 == 0; s++, t /= 2);
  wit = powmod(wit, t);
  if (wit == 1 || wit == n - 1) { return false; }
  for (int i = 1; i < s; ++i) {
    wit = mulmod(wit, wit);
    if (wit == 1) { return true; } if (wit == n - 1) { return false; }
  }
  return true;
}

// Is n prime?
template<class T> bool miller(T n) {
  if (n == 2) { return true; }
  if (n % 2 == 0 || n < 2) { return false; }
  // Jesli n > 2^32: losowac kilkanascie razy wit
  if (witness(T(2), n) || witness(T(7), n) || witness(T(61), n)) { return false; }
  return true;
}

// BABY STEP - GIANT STEP
// Returns dicrete log_a val (mod mod); -1 if no such exponent exists
LL disc_log(LL a, LL val, LL mod) {
  LL d = __gcd(a, mod);
  if(d != 1 && d != mod) { return -1; }
  LL sq = sqrt(mod) + 2;
  auto mulmod = [&](LL a, LL b) { return ((__int128_t)a * b) % mod; };
  auto powmod = [&](LL a, LL b) { return fastop(a, b, 1LL, mulmod); };
  map<LL, int> M;
  LL b = val, c = powmod(a, sq);
  for(int i = 0; i <= sq; ++i) {
    M[b] = i; b = mulmod(b, a);
  }
  b = c;
  for(int i = 1; i <= sq; ++i) {
    if (M.count(b)) { return i * sq - M[b]; }
    b = mulmod(b, c);
  }
  return -1;
}

// SOLOVAY-STRASSEN
// Computes jacobi symbol; m must be odd
// if m prime => result = 0 if m | n, 1 if n is a quadr. resid. mod m, -1 otherwise
```

```
template<class T> int jacobi(T n, T m) {
  int r = 1; n %= m;
  while (n > 1) {
    while (n % 2 == 0) {
      n /= 2;
      if (m % 8 == 3 || m % 8 == 5) { r *= -1; }
    }
    swap(n, m);
    if (n % 4 == 3 && m % 4 == 3) { r *= -1; }
    n %= m;
  }
  return r;
}

// POLARD'S RHO
// a - parameter, shall not be equal to 0 or -2.
// returns a divisor, a proper one when succeeded, equal to n if failed
// in case of failure, change a
template<class T> T rho(const T &n, const T a) {
  auto addmod = [&](const T &a, const T &b) { return (a + b) % n; };
  auto mulmod = [&](const T &a, const T &b) { return fastop(a, b, T(0), addmod); };
  auto f = [&](const T &x) { return addmod(mulmod(x, x), a); };
  T x = 2, y = 2;
  while (true) {
    x = f(x); y = f(f(y)); T d = __gcd(n, abs(x - y)); if (d != 1) { return d; }
  }
}

template<class T> T get_factor(T n) {
  if (n % 2 == 0) { return 2; } if (n % 3 == 0) { return 3; }
  if (n % 5 == 0) { return 5; }
  while (true) {
    T d = rho(n, T(rand()%100 + 2)); if(d != n) { return d; }
  }
}

template<class T> void __factorize(const T &n, vector<T> &x) {
  if (n == 1) { return; }
  else if (miller(n)) { x.push_back(n); }
  else {
    T d = get_factor(n); __factorize(d, x); __factorize(n / d, x);
  }
}

template<class T> vector<T> factorize(const T &n) {
  vector<T> x; __factorize(n, x); return x;
}

// TONELLI-SHANKS
// p must be prime, n must be a quadratic residue mod p (OTHERWISE IT LOOPS)
// returns a, such that a^2 mod p = n
template<class T> T sqrtmod(T n, T p) {
  auto addmod = [&](const T &a, const T &b) { return (a + b) % p; };
  auto mulmod = [&](const T &a, const T &b) { return fastop(a, b, T(0), addmod); };
  auto powmod = [&](const T &a, const T &b) { return fastop(a, b, T(1), mulmod); };
  int s; T q;
  for(s = 0, q = p - 1; q % 2 == 0; s++, q /= 2);
  if(s == 1) { return powmod(n, (p + 1) / 4); }
  LL z;
  do {
    z = rand() % (p - 1) + 1;
  } while(powmod(z, (p - 1) / 2) == 1);
  T c = powmod(T(z), q); T r = powmod(n, (q + 1) / 2);
  T t = powmod(n, q); T m = s;
  while(true) {
    if(t == 1) { return r; }
    int i;
    for(i = 1; powmod(t, T(1LL << i)) != 1; ++i);
    T b = powmod(c, T(1LL << (m - i - 1)));
    r = mulmod(r, b); c = mulmod(b, b);
    t = mulmod(t, c); m = i;
  }
```



```
}

template<class T> T primitive_root(T p) {
  vector<T> v = factorize(p - 1);
  v.erase(unique(ALL(v)), v.end());
  auto addmod = [&](const T &a, const T &b) { return (a + b) % p; };
  auto mulmod = [&](const T &a, const T &b) { return fastop(a, b, T(0), addmod); };
  auto powmod = [&](const T &a, const T &b) { return fastop(a, b, T(1), mulmod); };
  T x;
  do {
    x = rand() % (p - 1) + 1;
  } while (any_of(ALL(v), [&](const T &y) { return powmod(x, (p - 1) / y) == 1; }));
  return x;
}


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      PalindromicTree.hpp                                                 %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
const int ALFA = 30;
struct node{
  int dl;
  node* ds;
  node* d[ALFA];//zmienic na seta przy duzym alfabecie
  node(int dl) : dl(dl), ds(0) { R(i,ALFA) { d[i] = 0; } }
  ~node(){ R(i,ALFA) { if(d[i] != 0) { delete d[i]; } }
};
void buduj(int* a,int n){ // moza zmianic int* na char*
  node* os = new node(-1);
  node* korz = new node(0);
  korz->ds = os;
  R(i,n){
    while(i == os->dl || a[i] != a[i-os->dl-1]) os = os->ds;
    if(os->d[a[i]] == 0){
      os->d[a[i]] = new node(os->dl+2);
      if(os->dl == -1)
        os->d[a[i]]->ds = korz;
      else{
        node* pom = os->ds;
        while(pom->d[a[i]] == 0 || a[i]!=a[i-pom->dl-1]) { pom = pom->ds; }
        os->d[a[i]]->ds = pom->d[a[i]];
      }
    }
    os = os->d[a[i]];
  }
  delete korz->ds;
  delete korz;
}


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      Simplex.hpp                                                         %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "../Headery.hpp"

struct Simplex {          // Maximize c*x subject to Ax <= b.
  using T = double;       // Initialize the structure, set A, b, c and then run
  vector<vector<T>> A;    // solve(). Max objective is stored in res. To recover
  vector<T> b, c;         // the best result, use getVars().

  int V, E;
  VI eqIds, varIds, cols;
  T res;
  const T kEps = 1e-9;

  Simplex(int vars, int eqs) : A(eqs, vector<T>(vars)), b(eqs), c(vars),
                               V(vars), E(eqs), eqIds(eqs), varIds(vars), res(0) {
    iota(ALL(varIds), 0); iota(ALL(eqIds), vars);
  }

  void pivot(int eq, int var) {
    T coef = 1 / A[eq][var];
```

```
    cols.clear();
    REP (i, V) {
      if (abs(A[eq][i]) > kEps) { cols.PB(i); A[eq][i] *= coef; }
    }
    A[eq][var] *= coef; b[eq] *= coef;
    REP (row, E) {
      if (row == eq || abs(A[row][var]) < kEps) { continue; }
      T k = -A[row][var];
      A[row][var] = 0;
      for (int i : cols) { A[row][i] += k * A[eq][i]; }
      b[row] += k * b[eq];
    }
    T q = c[var]; c[var] = 0;
    for (int i : cols) { c[i] -= q * A[eq][i]; }
    res += q * b[eq];

    swap(varIds[var], eqIds[eq]);
  }

  bool solve() {
    while (true) {
      int eq = -1, var = -1;
      REP (i, E) { if (b[i] < -kEps) { eq = i; break; } }
      if (eq == -1) { break; }
      REP (i, V) { if (A[eq][i] < -kEps) { var = i; break; } }
      if (var == -1) { res = -1e9; return false; /* No solution */ }
      pivot(eq, var);
    }

    while (true) {
      int var = -1, eq = -1;
      REP (i, V) { if (c[i] > kEps) { var = i; break; } }
      if (var == -1) { break; }
      REP (i, E) {
        if (A[i][var] < kEps) { continue; }
        if (eq >= 0 && b[i] / A[i][var] >= b[eq] / A[eq][var]) { continue; }
        eq = i;
      }
      if (eq == -1) { res = 1e9; return false; /* Unbounded */ }
      pivot(eq, var);
    }

    return true;
  }

  vector<T> getVars() {  // Optimal assignment of variables.
    vector<T> result(V);
    REP (i, E) { if (eqIds[i] < V) { result[eqIds[i]] = b[i]; } }
    return result;
  }
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      SufArrLCP.hpp                                                       %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
struct SA {
  vector<pair<PII, int>> x;
  VI suf; // <- numery kolejnych sufiksow w porzadku leksykograficznym
  VI rank; // <- odwrotnosc tablicy sufiksowej (suf)
  VI lcp; // lcp[i] == lcp(suf[i-1],suf[i])
  int *z;
  int n;
  SA(int *z_, int n_) : z(z_), n(n_) {
    z[n] = -1; // straznik zaklada ze -1 nie wystepuje w z
    //ostroznie coby nic nie nadpisac
    suf.resize(n); rank.resize(n); x.resize(n);
    R(i, n) { x[i] = {{z[i], 0}, i}; }
    mapuj();
    int krok = 1;
    while (krok < n) {
      R (i, n) { x[i] = {{rank[i], i + krok < n ? rank[i + krok] : -1}, i}; }
      mapuj();
      krok *= 2;
    }
    R(i, n) { suf[rank[i]] = i; }

    //LCP - opcjonalnie
    lcp.resize(n);
    int k = 0;
    R(i, n) {
      int ak = rank[i];
      if(ak) { while (z[suf[ak] + k] == z[suf[ak - 1] + k]) k++; }
      lcp[rank[i]] = k;
      if(k) { k--; }
    }
    //koniec LCP
  };
  void mapuj() {
    sort(ALL(x));
    int id = 0;
    R (i, n) {
      if (i && x[i - 1].st != x[i].st) id++;
      rank[x[i].nd] = id;
    }
  }
};
                                 // 13 1 2 1 1 2 1 2 1 1 2 1 1 2
int z[(int)1e6], n;              //   -> 10 7 2 11 8 5 0 3 12 9 6 1 4
int32_t main(){                  //   -> 3 4 1 2 5 6 3 0 1 4 5 2
  scanf("%lld",&n);
  R(i,n)scanf("%lld",&z[i]);
  SA sa(z,n);
  R(i,n)printf("%lld ",sa.suf[i]);
  puts("");
  for(int i = 1; i < n; i++) printf("%lld ",sa.lcp[i]);
  puts("");
}


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      SuffixAutomation.hpp                                                %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "../Headery.hpp"

struct SuffixAutomaton {
  vector<map<char,int>> edges;
  VI link, length;
  int last;

  SuffixAutomaton(string s) : edges(1), link{-1}, length(1), last(0) {
    for (int i = 0; i < SZ(s); i++) {
      edges.PB(map<char,int>());
      length.PB(i+1);
      link.PB(0);
      int r = SZ(edges) - 1, p = last;
      while (p >= 0 && edges[p].find(s[i]) == edges[p].end()) {
        edges[p][s[i]] = r;
        p = link[p];
      }
      if (p != -1) {
        int q = edges[p][s[i]];
        if (length[p] + 1 == length[q]) {
          link[r] = q;
        } else {
          edges.PB(edges[q]);
          length.PB(length[p] + 1);
          link.PB(link[q]);
          int qq = SZ(edges)-1;
          link[q] = qq; link[r] = qq;
          while (p >= 0 && edges[p][s[i]] == q) {
            edges[p][s[i]] = qq;
            p = link[p];
```

```
        }
      }
    }
    last = r;
   }
  }
};


/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      SPFA.hpp                                                               %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
struct SPFA{
  int n; vector<int> odl, oj, czok;
  vector<vector<PII>> d; vector<vector<int>> d2;
  const int inf = 1e15;
  SPFA(int _n):n(_n+1){
    odl.resize(n, inf); oj.resize(n); czok.resize(n);
    d.resize(n); d2.resize(n);
  }
  vector<int> cykl; int root;
  bool us(int nr){
    if(nr == root) return 1;
    czok[nr] = 0;
    for(int ak:d2[nr]){
      if(oj[ak] == nr){
        if(us(ak)){
          cykl.PB(nr);
          return 1;
        }
      }
    }
    d2[nr].clear();
    return 0;
  }
  bool licz_sciezki(int s){ // false, gdy z s da sie dojsc do ujemnego cyklu
    vector<int> st, st2;    // znaleziony cykl jest w wektorze cykl
    odl[s] = 0; czok[s] = 1; st.PB(s);
    while(st.size()){
      R(i, st.size()){
        int ak = st[i];
        if(czok[ak]) for(PII x:d[ak]){
          int nei, cost; tie(nei, cost) = x;
          if(odl[ak] + cost < odl[nei]){
            root = ak;
            if(us(nei)){
              cykl.PB(ak); reverse(ALL(cykl));
              return 0;
            }
            odl[nei] = odl[ak] + cost; oj[nei] = ak;
            d2[ak].PB(nei); czok[nei] = 1;
            st2.PB(nei);
          }
        }
      }
      st.clear(); swap(st, st2);
    }
    return 1;
  }
  vector<int> ujemny_cykl(){
    R(i, n-1) add_edge(n-1, i, 0);
    if (licz_sciezki(n-1)){
      return {};
    } else {
      return cykl;
    }
  }
  void add_edge(int a, int b, int cost){
    d[a].PB({b, cost});
  }
};
```

University of Warsaw Page 24

```
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
/%%%      SSS.hpp                                                                %%%%%%/
/%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
#include "Headery.hpp"
struct SSS{
  vector<vector<int>> d,drev; vector<int> ord,ss,cz;
  int is,n;
  SSS(int _n) : n(_n) {
    d.resize(n); drev.resize(n);
    cz.resize(n); ss.resize(n, -1);
  }
  void add_edge(int a,int b){
    d[a].PB(b); drev[b].PB(a);
  }
  void dfs(int nr){
    if(cz[nr]) return;
    cz[nr] = 1;
    for(int ak:d[nr])
      dfs(ak);
    ord.PB(nr);
  }
  void dfs2(int nr,int s){
    if(ss[nr] != -1) return;
    ss[nr] = s;
    for(int ak:drev[nr])
      dfs2(ak,s);
  }
  void licz(){
    R(i,n) dfs(i);
    is = 0;
    reverse(ALL(ord));
    for(int el:ord){
      if(ss[el] == -1){
        dfs2(el, is);
        is++;
      }
    }
  }
  vector<vector<int>> stworzgraf() {
    vector<vector<int>> res(is);
    R(i,n)for(int ak:d[i]){
      if(ss[i] != ss[ak])
        res[ss[i]].PB(ss[ak]);
    }
    for(auto &el:res){
      sort(ALL(el));
      el.resize(unique(ALL(el)) - el.begin());
    }
    return res;
  }
};
```

$\int \sqrt{x^2+1}\,\mathrm{dx} = \frac{1}{2}\left(x\sqrt{x^2+1} + \operatorname{arcsinh} x\right) + c \quad (\operatorname{arcsinh} = \texttt{asinh})$

$\int \sqrt{1-x^2}\,\mathrm{dx} = \frac{1}{2}\left(x\sqrt{1-x^2} + \arcsin x\right) + c$

$\int \frac{1}{ax^2+bx+c}\,\mathrm{dx} = \frac{2}{\sqrt{4ac-b^2}} \arctan \frac{2ax+b}{\sqrt{4ac-b^2}} \qquad (\Delta < 0)$

$\int \frac{x}{ax^2+bx+c}\,\mathrm{dx} = \frac{1}{2a}\ln|ax^2+bx+c| - \frac{b}{2a}\int \frac{\mathrm{dx}}{ax^2+bx+c}$

$\int \tan x\,\mathrm{dx} = -\ln|\cos x| + c$

$(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}, \qquad (\arccos x)' = -\frac{1}{\sqrt{1-x^2}}$

$\texttt{tgamma}(\texttt{t}) = \Gamma(t) = \int_0^\infty e^{t-1}e^{-x}\,\mathrm{dx}$

$\frac{1}{\pi} = 0.31831,\ \pi^2 = 9.86960,\ \frac{1}{\pi^2} = 0.10132,\ \frac{1}{e} = 0.36788,\ \gamma = 0.577215664901532$

$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + O(n^{-4})$

$\ln n! = n\ln n - n + \frac{1}{2}\ln(2\pi n) + \frac{1}{12n} - \frac{1}{360n^3} + \frac{1}{1260n^5} - O(n^{-7})$

$c(n,k) = \left[{n \atop k}\right] = |s(n,k)| =$ liczba permutacji n-elementowych o k cyklach

$s(n,k) = s(n-1,k-1) - (n-1)\ s(n-1,k), \quad s(0,0) = 1, \quad s(n,0) = 0$

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

$S(n,k) = \left\{{n \atop k}\right\} =$ liczba podziałów n elementów na k niepustych zbiorów

$\left\{{n+1 \atop k}\right\} = k\left\{{n \atop k}\right\} + \left\{{n \atop k-1}\right\}$

$\sum_{k=0}^n S(n,k)(x)_k = x^n$

$\sum_{j=0}^n S(n,j)s(j,k) = \sum_{j=0}^n s(n,j)S(j,k) = [n=k]$

$\left\{{0 \atop 0}\right\} = 1, \quad \left\{{n \atop 0}\right\} = 0, \quad \left\{{0 \atop n}\right\} = 0$

Catalan: $c_n = \frac{1}{n+1}\binom{2n}{n} \qquad \sum_{j=0}^k \binom{m}{j}\binom{n-m}{k-j} = \binom{n}{k} \qquad \sum_{p=k}^n \left[{n \atop p}\right]\binom{p}{k} = \left[{n+1 \atop k+1}\right],$

$\sum_{p=k}^n \binom{n}{p}\left\{{p \atop k}\right\} = \left\{{n+1 \atop k+1}\right\}, \quad \sum_{p=0}^n \binom{n}{p}\left\{{p \atop l}\right\}\left\{{n-p \atop m}\right\} = \left\{{n \atop l+m}\right\}\binom{l+m}{l}$, analogicznie dla $\left[\ \right]$

University of Warsaw Page 25