Universidad Rey Juan Carlos

Teamto de Verano

Cristian Perez, Francisco Tortola, Sergio Salazar

Ada Byron 2021

2021

Graph (1)

BFS.java

Description: BFS-DFS
Time: $\mathcal{O}(E+V)$

17 lines

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

public class BFS {
    public static void bfs(int vertices, int start, ArrayList<
        Integer>[] graf) {
        boolean[] visitados = new boolean[vertices];
        Queue<Integer> cola = new LinkedList<>();
        cola.add(start);
        while (cola.size() > 0) {
            Integer pop = cola.remove();
            if (graf[pop] == null) continue;
            for (Integer k : graf[pop]) {
                if (!visitados[k]) {
                    cola.add(k);
                    visitados[k] = true;
                } } } } }
```

TarjanPuntosArticulacion.java

Description: Encontrar los puntos de articulacion de un grafo. (Puntos que al ser eliminados desconectan G)
Time: $\mathcal{O}(E+V)$

42 lines

```
public class TarjanPuntosArticulacion {
    private static int n; //Vertices
    private static ArrayList<Integer>[] graf;
    private static int[] dfs_low, dfs_num, parents,puntart;
    private static boolean[] visit;

    private static void art (int u, int t){
        visit[u]=true;
        dfs_num[u]=t;
        dfs_low[u]=t++;
        int children=0;
        for(Integer v: graf[u]){
            if(!visit[v]){
                children++;
                parents[v] = u;
                art(v,t);
                dfs_low[u]=Math.min(dfs_low[u], dfs_low[v]);
                if(parents[u] == -1 && children>1){
                    puntart[u]=t;
                }
                if(parents[u] != -1 && dfs_low[v]>=dfs_num[u]){
                    puntart[u]=t;
                }
            }
            else if(v!=parents[u]){
                dfs_low[u]=Math.min(dfs_low[u],dfs_num[v]);
            } } }

    public static void main(String[] args){
        dfs_low= new int[n];
        dfs_num= new int[n];
        parents=new int[n];
        Arrays.fill(parents,-1);
        puntart=new int[n];
        visit= new boolean[n];
        art(0,0);
        int puntosdearticulacion=0;
        for(int i=0;i<n;i++){
            if(puntart[i]!=0) puntosdearticulacion++;
        }
    }
}
```

Dikjstra.java

Description: Shortest Path en un grafo ponderado
Time: $\mathcal{O}(E * log(V))$

20 lines

```
public class Dikjstra {
    public static void Dikjstra(int nodos, int inicio){
        PriorityQueue<IntPair> pq = new PriorityQueue<>();
        pq.offer(new IntPair(0,inicio)); //offer==add
        int[] dist = new int[nodos];
        Arrays.fill(dist,1000000000);
        dist[inicio]=0;

        while(!pq.isEmpty()){
            IntPair top = pq.poll(); //poll==remove
            int distop=top.d;
            int vtop=top.v;
            if(distop > dist[vtop]) continue;
            for(IntPair aux: graf[vtop]){
                int disaux=aux.d;
                int vaux=aux.v;
                if(dist[vtop]+disaux >= dist[vaux]) continue;
                dist[vaux]=dist[vtop]+disaux;
                pq.offer(new IntPair(dist[vaux],vaux));
            } } } }
```

FloydWarshall.java

Description: Encontrar la minima distincia entre TODOS los pares de un grafo, el grafo debe estar descrito por su lista de adyacencia graf[][]
Time: $\mathcal{O}(V^3)$

11 lines

```
public class FloydWarshall {

    public static int[][] graf;

    public static void FW(int n){
        for(int k=0;k<n;k++)
            for(int i=0;i<n;i++)
                for(int j=0;j<n;j++)
                    graf[i][j] = Math.min(graf[i][j], graf[i][k
                        ]+graf[k][j]);
    }
}
```

TopologicalSort.java

Description: Orden en el que realizar n tareas si 1->2 implica que para hacer 2 hace falta hacer 1
Time: $\mathcal{O}(E+V)$

25 lines

```
public class TopologicalSort {
    public static int n; //vertices
    public static ArrayList<Integer> list;
    public static boolean visitados[];
    public static ArrayList<Integer>[] graf;

    public static void dfs_tps(int u){
        visitados[u]=true;
        for (Integer k : graf[u]) {
            if(!visitados[k]){
                dfs_tps(k);
            }
        }
        list.add(u+1);
    }

    public static void main(String[] args) {
        for(int i=0;i<n;i++){
            if(!visitados[i])
                dfs_tps(i);
        }
        //Recorrido en orden inverso
        for(int i=list.size()-1;i>=0;i--){
            System.out.println(list.get(i));
        } } }
```

StrongConnectedComponents.java

Description: u-v en la misma scc si existe un camino de u a v y viceversa
Time: $\mathcal{O}(E+V)$

55 lines

```
public class SCC {
    public static LinkedList<Integer> orden;
    public static ArrayList<Integer>[] graf ;
    public static int[] dfs_num;
    public static int[] dfs_low;
    public static boolean[] visited;
    public static int contador;
    public static int numSCC;

    public static int strongConnectedComponents(int u){
        dfs_low[u]=dfs_num[u]=contador++;
        orden.addLast(u);
        visited[u]=true;
  int size=0;
        for(int i=0;i<graf[u].size();i++){
            int v = graf[u].get(i);
            if(dfs_num[v]==-1){
                size=Math.max(strongConnectedComponents(v),size
                    );
            }
            if(visited[v])
                dfs_low[u]=Math.min(dfs_low[u],dfs_low[v]);
        }
        int auxsize=0;
        if(dfs_low[u]==dfs_num[u]){
            numSCC++;
            System.out.print("SCC "+numSCC+":");
            while(true){
                auxsize++;
                int v = orden.removeLast();
                visited[v]=false;
                System.out.print(v+" ");
                if(u==v) break;
            }
            System.out.println();
        }
  size=Math.max(size,auxsize)
        return size;
    }

    public static void main(String[] args) throws IOException {
        orden = new LinkedList<>();
        dfs_low=new int[h];
        dfs_num=new int[h];
        Arrays.fill(dfs_num,-1);
        Arrays.fill(dfs_low,-1);
        visited=new boolean[h]
        contador=0;
        numSCC=0;
        for(int i=0;i<V;i++){
            if(dfs_num[i]==-1){
                strongConnectedComponents(i);
            }
        }
    }
}
```

Mathematics (2)

2.1 Equations

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

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

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

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

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

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

2.2 Geometry

2.2.1 Quadrilaterals

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

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

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

2.2.2 Pick's theorem

$A = i + b/2 - 1$ Boundary point(b): a lattice point on the polygon (including vertices) Interior Point(i): a lattice point in the polygon's interior region

2.2.3 Volumes

	Sphere	Cube	Tetrahedron
Area	$4\pi r^2$	$6a^2$	$a^2\sqrt{3}$
Volume	$\frac{4}{3}\pi r^3$	a^3	$\frac{1}{12}a^3\sqrt{2}$

	Octahedron	Dodecahedron	Icosahedron
Area	$2\sqrt{3}a^2$	$3a^2\sqrt{25+10\sqrt{5}}$	$5\sqrt{3}a^2$
Volume	$\frac{1}{3}\sqrt{2}a^3$	$\frac{1}{4}(15+7\sqrt{5})a^3$	$\frac{5}{12}(3+\sqrt{5})a^3$

2.3 Sums

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

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

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

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

2.3.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

GCD/LCM

GCDLCM.cpp

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

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

5 lines

```
int gcd(int a, int b) {
while (b > 0) {
int temp = b; b = a % b; a = temp; }
return a; }
int lcm(int a, int b){ return a*(b/gcd(a,b));}
```

Geometry (3)

3.1 Geometric primitives

Point.h

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

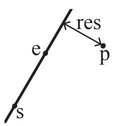
25 lines

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

lineDistance.h

Description:

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



"Point.h" 4 lines

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

pointsToLine.h

Description: Convert two points to Line

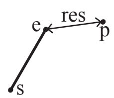
9 lines

```
// the answer is stored in the third parameter (pass by
    reference)
void pointsToLine(point p1, point p2, line &l) {
  if (fabs(p1.x - p2.x) < EPS) {  // vertical line is fine
    l.a = 1.0;   l.b = 0.0;   l.c = -p1.x;// default values
  } else {
    l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
    l.b = 1.0; // IMPORTANT: we fix the value of b to 1.0
    l.c = -(double)(l.a * p1.x) - p1.y;
} }
```

SegmentDistance.h

Description:

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



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

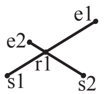
"Point.h" 6 lines

```
typedef Point<double> P;
```

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

SegmentIntersection.h

Description:
If a unique intersetion point between the line segments going from s1 to e1 and from s2 to e2 exists r1 is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists 2 is returned and r1 and r2 are set to the two ends of the common line. The wrong position will be returned if P is Point<int> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long. Use segmentIntersectionQ to get just a true/false answer.



Usage:
```
Point<double> intersection, dummy;
if (segmentIntersection(s1,e1,s2,e2,intersection,dummy)==1)
cout << "segments intersect at " << intersection << endl;
```

"Point.h" 27 lines

```
template <class P>
int segmentIntersection(const P& s1, const P& e1,
    const P& s2, const P& e2, P& r1, P& r2) {
  if (e1==s1) {
    if (e2==s2) {
      if (e1==e2) { r1 = e1; return 1; } //all equal
      else return 0; //different point segments
    } else return segmentIntersection(s2,e2,s1,e1,r1,r2);//swap
  }
  //segment directions and separation
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = v1.cross(d), a2 = v2.cross(d);
  if (a == 0) { //if parallel
    auto b1=s1.dot(v1), c1=e1.dot(v1),
         b2=s2.dot(v1), c2=e2.dot(v1);
    if (a1 || a2 || max(b1,min(b2,c2))>min(c1,max(b2,c2)))
      return 0;
    r1 = min(b2,c2)<b1 ? s1 : (b2<c2 ? s2 : e2);
    r2 = max(b2,c2)>c1 ? e1 : (b2>c2 ? s2 : e2);
    return 2-(r1==r2);
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  if (0<a1 || a<-a1 || 0<a2 || a<-a2)
    return 0;
  r1 = s1-v1*a2/a;
  return 1;
}
```

SegmentIntersectionQ.h

Description: Like segmentIntersection, but only returns true/false. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

"Point.h" 16 lines

```
template <class P>
bool segmentIntersectionQ(P s1, P e1, P s2, P e2) {
  if (e1 == s1) {
    if (e2 == s2) return e1 == e2;
    swap(s1,s2); swap(e1,e2);
  }
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = d.cross(v1), a2 = d.cross(v2);
  if (a == 0) { // parallel
    auto b1 = s1.dot(v1), c1 = e1.dot(v1),
         b2 = s2.dot(v1), c2 = e2.dot(v1);
    return !a1 && max(b1,min(b2,c2)) <= min(c1,max(b2,c2));
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  return (0 <= a1 && a1 <= a && 0 <= a2 && a2 <= a);
}
```

segmentIntersectionPoint.h

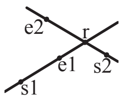
Description: Segment Intersection given the points

34 lines

```
double dist(point p1, point p2) {// Euclidean distance
  return hypot(p1.x - p2.x, p1.y - p2.y); }

struct vec { double x, y;
  vec(double _x, double _y) : x(_x), y(_y) {} };

vec toVec(point a, point b) {
  return vec(b.x - a.x, b.y - a.y); }

double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }

double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }

vec scale(vec v, double s) {  // nonnegative s = [<1 .. 1 ..
    >1]
  return vec(v.x * s, v.y * s); }// shorter.same.longer

point translate(point p, vec v) { // translate p according to v
  return point(p.x + v.x , p.y + v.y); }

double distToLine(point p, point a, point b, point &c) {
  // formula: c = a + u * ab
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  c = translate(a, scale(ab, u)); // translate a to c
  return dist(p, c); }

double distToLineSegment(point p, point a, point b, point &c) {
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  if (u < 0.0) { c = point(a.x, a.y);// closer to a
    return dist(p, a); }
  if (u > 1.0) { c = point(b.x, b.y); // closer to b
    return dist(p, b); }
  return distToLine(p, a, b, c); } // run distToLine as above
```

lineIntersection.h

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



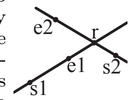
Usage:
```
point<double> intersection;
if (1 == LineIntersection(s1,e1,s2,e2,intersection))
cout << "intersection point at " << intersection << endl;
```

"Point.h" 9 lines

```
template <class P>
int lineIntersection(const P& s1, const P& e1, const P& s2,
    const P& e2, P& r) {
  if ((e1-s1).cross(e2-s2)) { //if not parallell
    r = s2-(e2-s2)*(e1-s1).cross(s2-s1)/(e1-s1).cross(e2-s2);
    return 1;
  } else
    return -((e1-s1).cross(s2-s1)==0 || s2==e2);
}
```

lineIntersectionv2.h

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



Usage:
```
point<double> intersection;
if (1 == LineIntersection(s1,e1,s2,e2,intersection))
cout << "intersection point at " << intersection << endl;
```

11 lines

```
struct line { double a, b, c; };

bool areIntersect(line l1, line l2, point &p) {
    int den = l1.a*l2.b - l1.b*l2.a;
    if(!den) return false; //same line or parallel
    p.x = l1.c*l2.b - l1.b*l2.c;
    if(p.x)  p.x /= den;
    p.y = l1.a*l2.c - l1.c*l2.a;
    if(p.y)  p.y /= den;
    return true;
}
```

sideOf.h

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

Usage: `bool left = sideOf(p1,p2,q)==1;`

"Point.h" 11 lines

```
template <class P>
int sideOf(const P& s, const P& e, const P& p) {
  auto a = (e-s).cross(p-s);
  return (a > 0) - (a < 0);
}
template <class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

onSegment.h

Description: Returns true iff p lies on the line segment from s to e. Intended for use with e.g. Point<long long> where overflow is an issue. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

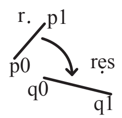
"Point.h" 5 lines

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

linearTransformation.h

Description:

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



"Point.h" 6 lines

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

```
}
```

Angle.h

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

37 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int quad() const {
    assert(x || y);
    if (y < 0) return (x >= 0) + 2;
    if (y > 0) return (x <= 0);
    return (x <= 0) * 2;
  }
  Angle t90() const { return {-y, x, t + (quad() == 3)}; }
  Angle t180() const { return {-x, -y, t + (quad() >= 2)}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.quad(), a.y * (ll)b.x) <
         make_tuple(b.t, b.quad(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

CanFormTriangle.cpp

Description: Return true if you can form a triangle
Time: $\mathcal{O}(1)$

2 lines

```
bool canFormTriangle(double a, double b, double c) {
  return (a + b > c) && (a + c > b) && (b + c > a); }
```

CanFormQuadrangle.cpp

Description: Return true if you can form a quadrangle
Time: $\mathcal{O}(1)$

2 lines

```
bool canFormQuadrangle(double a, double b, double c,double d) {
  return (a + b + c> d) && (a + c + d > b) && (b + c + d > a)
      && (a+b+d>c); }
```

hasIntersectQuadrangles.cpp

Description: Return true if two quadrangles intersect
Time: $\mathcal{O}(1)$

7 lines

```
int hasIntersectQuadrangles(int lx, int ly, int rx, int ry, int
      la, int lb, int ra, int rb) {
  lx = lxsol = max(lx, la);
  ly = lysol = max(ly, lb);
  rx = rxsol = min(rx, ra);
  ry = rysol = min(ry, rb);
  return lx < rx && ly < ry;
}
```

3.2 Circles

CircleIntersection.h

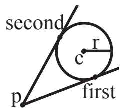
Description: Computes a pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" 14 lines

```
typedef Point<double> P;
bool circleIntersection(P a, P b, double r1, double r2,
    pair<P, P>* out) {
  P delta = b - a;
  assert(delta.x || delta.y || r1 != r2);
  if (!delta.x && !delta.y) return false;
  double r = r1 + r2, d2 = delta.dist2();
  double p = (d2 + r1*r1 - r2*r2) / (2.0 * d2);
  double h2 = r1*r1 - p*p*d2;
  if (d2 > r*r || h2 < 0) return false;
  P mid = a + delta*p, per = delta.perp() * sqrt(h2 / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

circleTangents.h

Description:



Returns a pair of the two points on the circle with radius r centered around c whos tangent lines intersect p. If p lies within the circle NaN-points are returned. P is intended to be Point<double>. The first point is the one to the right as seen from the p towards c.
Usage: `typedef Point<double> P; pair<P,P> p = circleTangents(P(100,2),P(0,0),2);`

"Point.h" 6 lines

```
template <class P>
pair<P,P> circleTangents(const P &p, const P &c, double r) {
  P a = p-c;
  double x = r*r/a.dist2(), y = sqrt(x-x*x);
  return make_pair(c+a*x+a.perp()*y, c+a*x-a.perp()*y);
}
```

inCircle.cpp

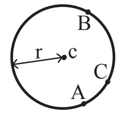
Description: Return incircle radio, area and perimeter of triangle.
Time: $\mathcal{O}(1)$

12 lines

```
double perimeter(double ab, double bc, double ca) {
  return ab + bc + ca; }

double area(double ab, double bc, double ca) {
  // Heron's formula
  double s = 0.5 * perimeter(ab, bc, ca);
  return sqrt(s) * sqrt(s - ab) * sqrt(s - bc) * sqrt(s - ca);
      }

double rInCircle(double ab, double bc, double ca) {
  double per = perimeter(ab, bc, ca);
  if(per==0) return 0;
  return area(ab, bc, ca) / (0.5 * per); }
```

circumcircle.h

Description:



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

"Point.h" 10 lines

```
#define PI acos(-1.0)
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h

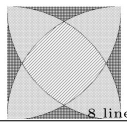
Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$

"circumcircle.h" 28 lines

```
pair<double, P> mec2(vector<P>& S, P a, P b, int n) {
  double hi = INFINITY, lo = -hi;
  FOR(i,0,n) {
    auto si = (b-a).cross(S[i]-a);
    if (si == 0) continue;
    P m = ccCenter(a, b, S[i]);
    auto cr = (b-a).cross(m-a);
    if (si < 0) hi = min(hi, cr);
    else lo = max(lo, cr);
  }
  double v = (0 < lo ? lo : hi < 0 ? hi : 0);
  P c = (a + b) / 2 + (b - a).perp() * v / (b - a).dist2();
  return {(a - c).dist2(), c};
}
pair<double, P> mec(vector<P>& S, P a, int n) {
  random_shuffle(S.begin(), S.begin() + n);
  P b = S[0], c = (a + b) / 2;
  double r = (a - c).dist2();
  FOR(i,1,n) if ((S[i] - c).dist2() > r * (1 + 1e-8)) {
    tie(r,c) = (n == sz(S) ?
      mec(S, S[i], i) : mec2(S, a, S[i], i));
  }
  return {r, c};
}
pair<double, P> enclosingCircle(vector<P> S) {
  assert(!S.empty()); auto r = mec(S, S[0], sz(S));
  return {sqrt(r.first), r.second};
}
```

QuarterCircles.cpp

Description:



Returns de Area of the QuarterCircles

8 lines

```
#define PI acos(-1)
double a;
double x, y, z;
z = a*a - a*a*PI/4;
z -= a*a*PI/4 - a*a*PI/6 - ( a*a*PI/6 - a*a*sqrt(3.0)/4 ); //
    outside
y = a*a - a*a*PI/4 - 2*z; // "triangles"
x = a*a - 4*y - 4*z; //middle
printf("%.3lf %.3lf %.3lf\n", x, 4*y ,4*z);
```

insideCircle.cpp

Description: Return points inside circle
Time: $\mathcal{O}(1)$

4 lines

```
int insideCircle(point p, point c, double r) { // all integer
     version
  double dx = p.x - c.x, dy = p.y - c.y;
  double Euc = dx * dx + dy * dy, rSq = r * r; // all integer
  return Euc < rSq ? 0 : Euc == rSq ? 1 : 2; } //inside/border/
      outside
```

circle2PtsRad.h

Description: Given 2 points on the circle (p1 and p2) and radius r of the corresponding circle, we can determine the location of the centers (c1 and c2) of the two possible circles

9 lines

```
bool circle2PtsRad(point p1, point p2, double r, point &c) {
  double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
              (p1.y - p2.y) * (p1.y - p2.y);
  double det = r * r / d2 - 0.25;
  if (det < 0.0) return false;
  double h = sqrt(det);
  c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
  c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
  return true; }
```

inCircumCircle.h

Description: Read below

25 lines

```
// returns 1 if there is a circumCenter center, returns 0
    otherwise
// if this function returns 1, ctr will be the circumCircle
    center
// and r is the same as rCircumCircle
int circumCircle(point p1, point p2, point p3, point &ctr,
    double &r){
  double a = p2.x - p1.x, b = p2.y - p1.y;
  double c = p3.x - p1.x, d = p3.y - p1.y;
  double e = a * (p1.x + p2.x) + b * (p1.y + p2.y);
  double f = c * (p1.x + p3.x) + d * (p1.y + p3.y);
  double g = 2.0 * (a * (p3.y - p2.y) - b * (p3.x - p2.x));
  if (fabs(g) < EPS) return 0;

  ctr.x = (d*e - b*f) / g;
  ctr.y = (a*f - c*e) / g;
  r = dist(p1, ctr);  // r = distance from center to 1 of the 3
      points
  return 1; }

// returns true if point d is inside the circumCircle defined
    by a,b,c
int inCircumCircle(point a, point b, point c, point d) {
  return (a.x - d.x) * (b.y - d.y) * ((c.x - d.x) * (c.x - d.x)
        + (c.y - d.y) * (c.y - d.y)) +
          (a.y - d.y) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y)
              * (b.y - d.y)) * (c.x - d.x) +
          ((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)
             ) * (b.x - d.x) * (c.y - d.y) -
          ((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)
             ) * (b.y - d.y) * (c.x - d.x) -
          (a.y - d.y) * (b.x - d.x) * ((c.x - d.x) * (c.x - d.x)
              + (c.y - d.y) * (c.y - d.y)) -
          (a.x - d.x) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y)
              * (b.y - d.y)) * (c.y - d.y) > 0 ? 1 : 0;
}
```

3.3 Polygons

insidePolygon.h

Description: Returns true if p lies within the polygon described by the points between iterators begin and end. If strict false is returned when p is on the edge of the polygon. Answer is calculated by counting the number of intersections between the polygon and a line going from p to infinity in the positive x-direction. The algorithm uses products in intermediate steps so watch out for overflow. If points within epsilon from an edge should be considered as on the edge replace the line "if (onSegment..." with the comment bellow it (this will cause overflow for int and long long).
Usage: `typedef Point<int> pi;`
`vector<pi> v; v.push_back(pi(4,4));`
`v.push_back(pi(1,2)); v.push_back(pi(2,1));`
`bool in = insidePolygon(v.begin(),v.end(), pi(3,4), false);`
Time: $\mathcal{O}(n)$

"Point.h", "onSegment.h", "SegmentDistance.h" 14 lines

```
template <class It, class P>
bool insidePolygon(It begin, It end, const P& p,
    bool strict = true) {
  int n = 0; //number of isects with line from p to (inf,p.y)
  for (It i = begin, j = end-1; i != end; j = i++) {
    //if p is on edge of polygon
    if (onSegment(*i, *j, p)) return !strict;
    //or: if (segDist(*i, *j, p) <= epsilon) return !strict;
    //increment n if segment intersects line from p
    n += (max(i->y,j->y) > p.y && min(i->y,j->y) <= p.y &&
        ((*j-*i).cross(p-*i) > 0) == (i->y <= p.y));
  }
  return n&1; //inside if odd number of intersections
}
```

PolygonArea.h

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

"Point.h" 6 lines

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

PolygonAreav2.h

Description: Return polygon area.

6 lines

```
double calc_area(vector<P<double>> Pa) {
    double ans = 0;
    for(int i = 0; i < (int)Pa.size()-1; i++)
        ans += Pa[i].x*Pa[i+1].y - Pa[i].y*Pa[i+1].x;
    return fabs(ans)/2.0;
}
```

PolygonCenter.h

Description: Returns the center of mass for a polygon.

"Point.h" 10 lines

```
typedef Point<double> P;
Point<double> polygonCenter(vector<P>& v) {
  auto i = v.begin(), end = v.end(), j = end-1;
  Point<double> res{0,0}; double A = 0;
  for (; i != end; j=i++) {
    res = res + (*i + *j) * j->cross(*i);
    A += j->cross(*i);
  }
  return res / A / 3;
}
```

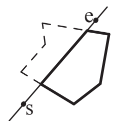
PolygonCenterOfMass.h

Description: Return center of mass

18 lines

```
template <class P>
P centroid(vector<P> g)   //center of mass
{
    double cx = 0.0, cy = 0.0;
    for(unsigned int i = 0; i < g.size() - 1; i++)
    {
        double x1 = g[i].x, y1 = g[i].y;
        double x2 = g[i+1].x, y2 = g[i+1].y;

        double f = x1 * y2 - x2 * y1;
        cx += (x1 + x2) * f;
        cy += (y1 + y2) * f;
    }
    double res = calc_area(g);    //remove abs
    cx /= 6.0 * res;
    cy /= 6.0 * res;
    return P(cx, cy);
}
```

PolygonCut.h



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

"Point.h", "lineIntersection.h" 15 lines

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

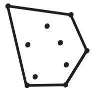
PolygonConvex.cpp

Description: Returns if the polygon it is convex
Time: $\mathcal{O}(N)$

16 lines

```
double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }
// note: to accept collinear points, we have to change the > 0
// returns true if point r is on the left side of line pq
bool ccw(point p, point q, point r) {
  return cross(toVec(p, q), toVec(p, r)) > 0; }
// returns true if point r is on the same line as the line pq
bool collinear(point p, point q, point r) {
  return fabs(cross(toVec(p, q), toVec(p, r))) < EPS; }
bool isConvex(const vector<point> &P) {
  int sz = (int)P.size();
  if (sz <= 3) return false;
  bool isLeft = ccw(P[0], P[1], P[2]);
  for (int i = 1; i < sz-1; i++)
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft)
      return false;
  return true; }
```

ConvexHull.h



Description: Returns a vector of indices of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Usage: `vector<P> ps, hull;`
`trav(i, convexHull(ps)) hull.push_back(ps[i]);`
Time: $\mathcal{O}(n \log n)$

"Point.h" 20 lines

```
typedef Point<ll> P;
pair<vi, vi> ulHull(const vector<P>& S) {
  vi Q(sz(S)), U, L;
  iota(all(Q), 0);
  sort(all(Q), [&S](int a, int b){ return S[a] < S[b]; });
  trav(it, Q) {
#define ADDP(C, cmp) while (sz(C) > 1 && S[C[sz(C)-2]].cross(\
  S[it], S[C.back()]) cmp 0) C.pop_back(); C.push_back(it);
    ADDP(U, <=); ADDP(L, >=);
  }
  return {U, L};
}

vi convexHull(const vector<P>& S) {
  vi u, l; tie(u, l) = ulHull(S);
  if (sz(S) <= 1) return u;
  if (S[u[0]] == S[u[1]]) return {0};
  l.insert(l.end(), u.rbegin()+1, u.rend()-1);
  return l;
}
```

newConvexHull.cpp

Description: Return a vector with the convexhull / convert to array if TLE
Time: $\mathcal{O}(Nlog(N))$

20 lines

```
template <class P>
double cross(P o, P a, P b) {
    return (a.x-o.x)*(b.y-o.y) - (a.y-o.y)*(b.x-o.x);

template <class P>
vector<P> CH(vector<P> Pa){
    vector<P> res;
    sort(Pa.begin(),Pa.end());
    int n = Pa.size();
    int m=0;
    for (int i=0;i<n;i++){
        while (m>1&&cross(res[m-2], res[m-1], Pa[i]) <= 0)res.
             pop_back(),m--;
        res.push_back(Pa[i]),m++;
    }
    for (int i = n-1, t = m+1; i >= 0; i--) {
        while (m>=t&&cross(res[m-2], res[m-1], Pa[i]) <= 0)res.
             pop_back(),m--;
        res.push_back(Pa[i]),m++;
    }
    return res;
}
```

PolygonPerimeter.cpp

Description: Returns the perimeter of a polygon
Time: $\mathcal{O}(N)$

10 lines

```
template <class P>
double dist(P p1, P p2) {
  return hypot(p1.x - p2.x, p1.y - p2.y); }

template <class P>
double perimeter(const vector<P> &Pa) {
  double result = 0.0;
  for (int i = 0; i < (int)Pa.size()-1; i++)
    result += dist(Pa[i], Pa[i+1]);
  return result; }
```

PolygonDiameter.h

Description: Calculates the max squared distance of a set of points.

"ConvexHull.h" 19 lines

```
vector<pii> antipodal(const vector<P>& S, vi& U, vi& L) {
  vector<pii> ret;
  int i = 0, j = sz(L) - 1;
  while (i < sz(U) - 1 || j > 0) {
    ret.emplace_back(U[i], L[j]);
    if (j == 0 || (i != sz(U)-1 && (S[L[j]] - S[L[j-1]])
          .cross(S[U[i+1]] - S[U[i]]) > 0)) ++i;
    else --j;
  }
  return ret;
}

pii polygonDiameter(const vector<P>& S) {
  vi U, L; tie(U, L) = ulHull(S);
  pair<ll, pii> ans;
  trav(x, antipodal(S, U, L))
    ans = max(ans, {(S[x.first] - S[x.second]).dist2(), x});
  return ans.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a given polygon (counter-clockwise order). The polygon must be such that every point on the circumference is visible from the first point in the vector. It returns 0 for points outside, 1 for points on the circumference, and 2 for points inside.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "onSegment.h" 22 lines

```
typedef Point<ll> P;
int insideHull2(const vector<P>& H, int L, int R, const P& p) {
  int len = R - L;
  if (len == 2) {
    int sa = sideOf(H[0], H[L], p);
    int sb = sideOf(H[L], H[L+1], p);
    int sc = sideOf(H[L+1], H[0], p);
    if (sa < 0 || sb < 0 || sc < 0) return 0;
    if (sb==0 || (sa==0 && L == 1) || (sc == 0 && R == sz(H)))
      return 1;
    return 2;
  }
  int mid = L + len / 2;
  if (sideOf(H[0], H[mid], p) >= 0)
    return insideHull2(H, mid, R, p);
  return insideHull2(H, L, mid+1, p);
}

int insideHull(const vector<P>& hull, const P& p) {
  if (sz(hull) < 3) return onSegment(hull[0], hull.back(), p);
  else return insideHull2(hull, 1, sz(hull), p);
}
```

LineHullIntersection.h

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

"Point.h" 63 lines

```
ll sgn(ll a) { return (a > 0) - (a < 0); }
typedef Point<ll> P;
struct HullIntersection {
  int N;
  vector<P> p;
  vector<pair<P, int>> a;

  HullIntersection(const vector<P>& ps) : N(sz(ps)), p(ps) {
    p.insert(p.end(), all(ps));
    int b = 0;
    FOR(i,1,N) if (P{p[i].y,p[i].x} < P{p[b].y, p[b].x}) b = i;
    FOR(i,0,N) {
      int f = (i + b) % N;
      a.emplace_back(p[f+1] - p[f], f);
    }
  }

  int qd(P p) {
    return (p.y < 0) ? (p.x >= 0) + 2
         : (p.x <= 0) * (1 + (p.y <= 0));
  }

  int bs(P dir) {
    int lo = -1, hi = N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (make_pair(qd(dir), dir.y * a[mid].first.x) <
        make_pair(qd(a[mid].first), dir.x * a[mid].first.y))
        hi = mid;
      else lo = mid;
    }
    return a[hi%N].second;
  }

  bool isign(P a, P b, int x, int y, int s) {
    return sgn(a.cross(p[x], b)) * sgn(a.cross(p[y], b)) == s;
  }

  int bs2(int lo, int hi, P a, P b) {
    int L = lo;
    if (hi < lo) hi += N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (isign(a, b, mid, L, -1)) hi = mid;
      else lo = mid;
    }
    return lo;
  }

  pii isct(P a, P b) {
    int f = bs(a - b), j = bs(b - a);
    if (isign(a, b, f, j, 1)) return {-1, -1};
    int x = bs2(f, j, a, b)%N,
        y = bs2(j, f, a, b)%N;
    if (a.cross(p[x], b) == 0 &&
        a.cross(p[x+1], b) == 0) return {x, x};
    if (a.cross(p[y], b) == 0 &&
        a.cross(p[y+1], b) == 0) return {y, y};
    if (a.cross(p[f], b) == 0) return {f, -1};
    if (a.cross(p[j], b) == 0) return {j, -1};
    return {x, y};
  }
};
```

3.4 Misc. Point Set Problems

closestPair.h

Description: $i1$, $i2$ are the indices to the closest pair of points in the point vector p after the call. The distance is returned.

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

"Point.h" 58 lines

```
template <class It>
bool it_less(const It& i, const It& j) { return *i < *j; }
template <class It>
bool y_it_less(const It& i,const It& j) {return i->y < j->y;}

template<class It, class IIt> /* IIt = vector<It>::iterator */
double cp_sub(IIt ya, IIt yaend, IIt xa, It &i1, It &i2) {
  typedef typename iterator_traits<It>::value_type P;
  int n = yaend-ya, split = n/2;
  if(n <= 3) { // base case
    double a = (*xa[1]-*xa[0]).dist(), b = 1e50, c = 1e50;
    if(n==3) b=(*xa[2]-*xa[0]).dist(), c=(*xa[2]-*xa[1]).dist()
        ;
    if(a <= b) { i1 = xa[1];
      if(a <= c) return i2 = xa[0], a;
      else return i2 = xa[2], c;
    } else { i1 = xa[2];
      if(b <= c) return i2 = xa[0], b;
      else return i2 = xa[1], c;
  } }
  vector<It> ly, ry, stripy;
  P splitp = *xa[split];
  double splitx = splitp.x;
  for(IIt i = ya; i != yaend; ++i) { // Divide
    if(*i != xa[split] && (**i-splitp).dist2() < 1e-12)
      return i1 = *i, i2 = xa[split], 0;// nasty special case!
    if (**i < splitp) ly.push_back(*i);
    else ry.push_back(*i);
  } // assert((signed)lefty.size() == split)
  It j1, j2; // Conquer
  double a = cp_sub(ly.begin(), ly.end(), xa, i1, i2);
  double b = cp_sub(ry.begin(), ry.end(), xa+split, j1, j2);
  if(b < a) a = b, i1 = j1, i2 = j2;
  double a2 = a*a;
  for(IIt i = ya; i != yaend; ++i) { // Create strip (y-sorted)
    double x = (*i)->x;
    if(x >= splitx-a && x <= splitx+a) stripy.push_back(*i);
  }
  for(IIt i = stripy.begin(); i != stripy.end(); ++i) {
    const P &p1 = **i;
    for(IIt j = i+1; j != stripy.end(); ++j) {
      const P &p2 = **j;
      if(p2.y-p1.y > a) break;
      double d2 = (p2-p1).dist2();
      if(d2 < a2) i1 = *i, i2 = *j, a2 = d2;
  } }
  return sqrt(a2);
}

template<class It> // It is random access iterators of point<T>
double closestpair(It begin, It end, It &i1, It &i2 ) {
  vector<It> xa, ya;
  assert(end-begin >= 2);
  for (It i = begin; i != end; ++i)
    xa.push_back(i), ya.push_back(i);
  sort(xa.begin(), xa.end(), it_less<It>);
  sort(ya.begin(), ya.end(), y_it_less<It>);
  return cp_sub(ya.begin(), ya.end(), xa.begin(), i1, i2);
}
```

HeronWithMedians.cpp

Description: Heron Theorem with medians 4/3

12 lines

```
int main()
{
    double d1,d2,d3;
    while(scanf("%lf%lf%lf",&d1,&d2,&d3)==3)
    {
        double res = (d1+d2+d3)/2;
        res=(res-d1)*(res-d2)*(res-d3)*res;
        if(res<EPS) printf("-1.000\n");
        else printf("%.3lf\n",sqrt(res)*4/3);
    }
    return 0;
}
```

3.5 3D

PolyhedronVolume.h

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

6 lines

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

Point3D.h

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

32 lines

```
template <class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

sphericalDistance.h

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

8 lines

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

Numerical (4)

Polynomial.h

17 lines

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

PolyRoots.h

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

"Polynomial.h" 23 lines

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

PolyInterpolate.h

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

13 lines

```
typedef vector<double> vd;
```

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

MatrixInverse.h

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

```
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  FOR(i,0,n) tmp[i][i] = 1, col[i] = i;

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

  for (int i = n-1; i > 0; --i) FOR(j,0,i) {
    double v = A[j][i];
    FOR(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }

  FOR(i,0,n) FOR(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}(n^2 m)$ 39 lines

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = A.size(), m = x.size(), rank = 0, br, bc;
  if (n) assert(A[0].size() == m);
  // FOR(i, 0, n) FOR(j, 0, m) A[i][j] %= MOD; also b[i]...
  vi col(m); iota(col.begin(), col.end(), 0);

  FOR(i,0,n) {
    double v, bv = 0;
    FOR(r,i,n) FOR(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      FOR(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    FOR(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    FOR(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      FOR(k,i,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    FOR(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

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

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

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$ 33 lines

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

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

Number theory (5)

5.1 Primality

eratosthenes.h

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

```
const int MAX_PR = 5000000;
bitset<MAX_PR> isprime;
vi eratosthenes_sieve(int lim) {
  isprime.set(); isprime[0] = isprime[1] = 0;
  for (int i = 4; i < lim; i += 2) isprime[i] = 0;
  for (int i = 3; i*i < lim; i += 2) if (isprime[i])
    for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
  vi pr;
  FOR(i,2,lim) if (isprime[i]) pr.push_back(i);
  return pr;
}
```

MillerRabin.h

Description: Miller-Rabin primality probabilistic test. Probability of failing one iteration is at most 1/4. 15 iterations should be enough for 50-bit numbers.
Time: 15 times the complexity of $a^b \mod c$.
"ModMulLL.h" 16 lines

```
bool prime(ull p) {
  if (p == 2) return true;
  if (p == 1 || p % 2 == 0) return false;
  ull s = p - 1;
  while (s % 2 == 0) s /= 2;
  FOR(i,0,15) {
    ull a = rand() % (p - 1) + 1, tmp = s;
    ull mod = mod_pow(a, tmp, p);
    while (tmp != p - 1 && mod != 1 && mod != p - 1) {
      mod = mod_mul(mod, mod, p);
      tmp *= 2;
    }
    if (mod != p - 1 && tmp % 2 == 0) return false;
  }
  return true;
}
```

factor.h

Description: Pollard's rho algorithm. It is a probabilistic factorisation algorithm, whose expected time complexity is good. Before you start using it, run `init(bits)`, where bits is the length of the numbers you use. to get factor multiple times, uncomment comments with (*)
Time: Expected running time should be good enough for 50-bit numbers.
"MillerRabin.h", "eratosthenes.h", "euclid.h" 37 lines

```
vector<ull> pr;
ull f(ull a, ull n, ull &has) {
  return (mod_mul(a, a, n) + has) % n;
}
vector<ull> factor(ull d) {
  vector<ull> res;
  for (size_t i = 0; i < pr.size() && pr[i]*pr[i] <= d; i++)
    if (d % pr[i] == 0) {
      while (d % pr[i] == 0) /*{ */ d /= pr[i];
      res.push_back(pr[i]); /*} (*)*/
    }
  //d is now a product of at most 2 primes.
  if (d > 1) {
    if (prime(d))
      res.push_back(d);
    else while (true) {
      ull has = rand() % 2321 + 47;
      ull x = 2, y = 2, c = 1;
      for (; c==1; c = gcd((y > x ? y - x : x - y), d)) {
        x = f(x, d, has);
        y = f(f(y, d, has), d, has);
      }
      if (c != d) {
        res.push_back(c); d /= c;
        if (d != c /* || true (*)*/) res.push_back(d);
        break;
      }
    }
  }
  return res;
}
void init(int bits) {//how many bits do we use?
  vi p = eratosthenes_sieve(1 << ((bits + 2) / 3));
  pr.resize(p.size());
  for (size_t i=0; i<pr.size(); i++)
    pr[i] = p[i];
}
```

5.2 Divisibility

euclid.h

Description: Finds the Greatest Common Divisor to the integers a and b. Euclid also finds two integers x and y, such that $ax + by = \gcd(a, b)$. If a and b are coprime, then x is the inverse of $a \pmod b$.

7 lines

```
ll gcd(ll a, ll b) { return __gcd(a, b); }

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

5.3 Primes

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

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

5.4 Estimates

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

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

Combinatorial (6)

6.1 Partitions and subsets

6.1.1 Partition function

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

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

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

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

6.1.2 Binomials

binomialModPrime.h

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

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

10 lines

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

6.2 General purpose numbers

6.2.1 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

6.2.2 Eulerian numbers

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

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

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

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

6.2.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.2.4 Bell numbers

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

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

6.2.5 Catalan numbers

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

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

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

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

Others (7)

MergeIntervals.java

Description: Union de Intervalos

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

28 lines

```
public class MergeIntervals {
    public static void main(String[] args) throws IOException {
        ArrayList<IntPair> al = new ArrayList<>();
        for(int i=0;i<q;i++){ //Extremos de los intervalos
            al.add(new IntPair(Integer.parseInt(st.nextToken())
                    ,Integer.parseInt(st.nextToken())));
        } //Ordenar de menor a mayor por el incio del intervalo
```

```
        Collections.sort(al);
        Stack<IntPair> stack = new Stack<>();
        stack.push(al.get(0));
        for(int i=1;i<al.size();i++){
            IntPair top = stack.peek();
            if(top.fini<al.get(i).ini){
                stack.push(al.get(i));
            }
            else if(top.fini<al.get(i).fini){
                top.fini=al.get(i).fini;
                stack.pop();
                stack.push(top);
            }
        }
        int total=0;
        while(!stack.isEmpty()){
            IntPair t= stack.pop();
            total+=(t.fini-t.ini+1);
        }
        System.out.println(total);
    }
}
```

Skyline.java

Description: Dado n edficios encontrar la forma/area del skyline, se devuelven los vertices superior izquierdo hasta el ultimo, que es inferior derecho
Time: $\mathcal{O}(nLog(n))$

110 lines

```
public class Skyline {
    public static List<IntPair> getSkyline(long[][] buildings)
     {
        int n = buildings.length;
        List<IntPair> salida = new ArrayList<IntPair>();

        if (n == 0) return salida;
        if (n == 1) {
            long xStart = buildings[0][0];
            long xEnd = buildings[0][1];
            long y = buildings[0][2];

            salida.add(new IntPair(xStart,y));
            salida.add(new IntPair(xEnd,0));
            return salida;
        }

        List<IntPair> leftSkyline, rightSkyline;
        leftSkyline = getSkyline(Arrays.copyOfRange(buildings,
             0, n / 2));
        rightSkyline = getSkyline(Arrays.copyOfRange(buildings,
              n / 2, n));

        return mergeSkylines(leftSkyline, rightSkyline);
    }

    public static List<IntPair> mergeSkylines(List<IntPair>
         left, List<IntPair> right) {
        long nL = left.size(), nR = right.size();
        int pL = 0, pR = 0;
        long currY = 0, leftY = 0, rightY = 0;
        long x, maxY;
        ArrayList<IntPair> salida = new ArrayList<IntPair>();
        while ((pL < nL) && (pR < nR)) {
            IntPair pointL = left.get(pL);
            IntPair pointR = right.get(pR);
            if (pointL.ini < pointR.ini) {
                x = pointL.ini;
                leftY = pointL.alt;
                pL++;
            }
            else {
                x = pointR.ini;
                rightY = pointR.alt;
                pR++;
            }
            maxY = Math.max(leftY, rightY);
            if (currY != maxY) {
                updateOutput(salida, x, maxY);
                currY = maxY;
            }
        }
        appendSkyline(salida, left, pL, nL, currY);
        appendSkyline(salida, right, pR, nR, currY);
        return salida;
    }

    public static void updateOutput(List<IntPair> output, long
         x, long y) {
        if (output.isEmpty() || output.get(output.size() - 1).
             ini != x)
            output.add(new IntPair(x,y));
        else {
            output.get(output.size() - 1).setAlt(y);
        }
    }

    public static void appendSkyline(List<IntPair> output, List
         <IntPair> skyline, int p, long n, long currY) {
        while (p < n) {
            IntPair point = skyline.get(p);
            long x = point.ini;
            long y = point.alt;
            p++;
            if (currY != y) {
                updateOutput(output, x, y);
                currY = y;
            }
        }
    }

    public static void main(String[] args) {
            long [][] skyline = new long[q][3];
            //0 ->Inicio 1->Final 2->Ancho
            List<IntPair> sl = getSkyline(skyline);
            long area=0;
            for(int j=0;j<sl.size()-1;j++){
                long a = sl.get(j).ini;
                long alt = sl.get(j).alt;
                long b = sl.get(j + 1).ini;
                area+=((b-a)*alt);
            }
            System.out.println(area);

        }
    }

    public static class IntPair implements Comparable{
        long ini;
        long alt;

        public IntPair(long i, long a){
            ini=i;
            alt=a;
        }

        public void setAlt(long alt) {
            this.alt = alt;
        }

        @Override
        public int compareTo(Object o) {
            IntPair i = (IntPair) o;
            return (int) (this.ini-i.ini);
        }
    }
}
```

Y si no ac? (8)

troubleshoot.txt

56 lines

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

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

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

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

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