#define MAXD 2

#define FOR(i,b,c) for(int i = b ; i < c ; i++ )

#define eps 1e-9

#define pi acos(-1.0)

#define SQ(x) ((x)\*(x))

**double cosineRule3Side ( double a, double b, double c ) {**

double res = (SQ(a)+SQ(b)-SQ(c)) / (2\*a\*b);

if ( res < -1 ) res = -1;

if ( res > 1 ) res = 1;

return acos ( res );

**}**

**struct myVec {**

int d; //Dimension

double val[MAXD];//Contains value of each component

**myVec() {**

d=2;

memset(val,0,sizeof(val));

**}**

**myVec(double \_x,double \_y) {**

d=2;

val[0]=\_x;

val[1]=\_y;

**}**

**myVec(int \_d,double\*a) {**

d=\_d;

for(int i=0; i<d; i++) {

val[i]=a[i];

}

**}**

**myVec add ( myVec b ) {**

myVec res;

for(int i=0; i<d; i++) res.val[i] = val[i] + b.val[i];

return res;

**}**

**myVec sub ( myVec b ) {**

myVec res;

for(int i=0; i<d; i++) res.val[i] = val[i] - b.val[i];

return res;

**}**

**myVec mul ( double t ) {**

myVec res;

for(int i=0; i<d; i++)res.val[i] = val[i] \* t;

return res;

**}**

**myVec div ( double t ) {**

myVec res;

for(int i=0; i<d; i++) res.val[i] = val[i] / t;

return res;

**}**

**bool operator == ( myVec b ) {**

for(int i=0; i<d; i++) if ( fabs ( val[i] - b.val[i] ) > eps ) return false;

return true;

**}**

**myVec perp2D() {**

myVec res = (\*this);

swap ( res.val[0], res.val[1] );

res.val[0] \*= -1;

return res;

**}**

**double dot ( myVec v ) { //Finds \*this (dot) v**

double res = 0;

for ( int i = 0; i < d; i++ ) res += val[i] \* v.val[i];

return res;

**}**

**double length () { //Finds length of current vector**

return sqrt ( this->dot( \*this ) );

**}**

**myVec unitVec () {**

return (\*this).div ( length() ); // v / ||v||

**}**

**double angleBetween2D ( Vec b ) {**

double pol1 = atan2 ( val[1], val[0] );

double pol2 = atan2 ( b.val[1], b.val[0] );

if ( pol2 + eps < pol1 ) pol2 += 2 \* pi;

double x = pol2 - pol1;

if ( x > pi + eps ) x = (2\*pi) - x;

//For direction, use sign of cross2D

return x;

**}**

//Causes precision error. Use angleBetween2D when 2D.

**double angleBetween ( myVec b ) { //Angle between two vectors**

double res = dot( b ) / ( length() \* b.length() );

if ( res > 1 ) res = 1;

if ( res < -1 ) res = -1;

return acos (res);

**}**

**double polarAngle2D() { //Angle from x-axis**

double res = atan2 ( val[1], val[0] );

if ( res + eps < 0 ) res += 2 \* pi;

return res;

**}**

**double cross2D ( myVec v ) { //Cross the two values. Only for 2D. Z compo 0.**

return val[0]\*v.val[1] - val[1]\*v.val[0];

**}**

//Provided, a comes before b. Otherwise, need to swap

**bool between ( Vec a, Vec b ) {**

if ( val[0] + eps < a.val[0] || val[0] > b.val[0] + eps ) return false;

if ( val[1] + eps < a.val[1] || val[1] > b.val[1] + eps ) return false;

return true;

**}**

**};**

**double triangleArea ( Vec a, Vec b, Vec c ) {**

double area = a.val[0] \* b.val[1] + b.val[0] \* c.val[1] + c.val[0] \* a.val[1];

area -= b.val[0] \* a.val[1] + c.val[0] \* b.val[1] + a.val[0] \* c.val[1];

area /= 2;

return area;

**}**

**double cosineRule3Side ( double a, double b, double c ) {**

double res = (SQ(a)+SQ(b)-SQ(c)) / (2\*a\*b);

if ( res < -1 ) res = -1; if ( res > 1 ) res = 1;

return acos ( res );

**}**

**struct myLine {**

myVec a, b; //a is displacement, b is direction.

//Builds a line from two points

**myLine() {}**

**myLine(myVec x, myVec y) {**

a=x;b=y.sub(x);

**}**

**myLine lineFromPoints ( myVec x, myVec y ) {**

myLine m;

m.a = x;

m.b = y.sub ( x );

return m;

**}**

//Finds point on line, given t.

**myVec atPos ( double t ) {**

return a.add ( b.mul ( t ) ); // a + tb;

**}**

**double lineToPointDistance ( myVec p, double t ) {**

p = p.sub ( a ); //Take it to origin

t = b.dot ( p ) / ( b.length() \* b.length() ); //point of intersection

myVec x = b.mul ( t ); //tb

return ( p.sub(x).length() ); //xp length()

**}**

**double segmentToPointDistance ( myVec p, double &t ) {**

p = p.sub ( a ); //Take it to origin

t = b.dot ( p ) / ( b.length() \* b.length() );

if ( t + eps < 0 || t > 1 + eps ) { //Not on segment

return min ( p.length(), p.sub(b).length() );

}

myVec x = b.mul ( t ); //tb

return ( p.sub(x).length() ); //xp length()

**}**

**bool overlapParallel ( myLine l ) {**

double p, q, r, s;

if ( b.val[0] == 0 ) {

p = a.val[1];

q = atPos(1).val[1];

r = l.a.val[1];

s = l.atPos ( 1 ).val[1];

if ( min ( r, s ) > max ( p, q ) ) return false;

if ( max ( r, s ) < min ( p, q ) ) return false;

return true;

} else {

p = a.val[0];

q = atPos(1).val[0];

r = l.a.val[0];

s = l.atPos ( 1 ).val[0];

if ( min ( r, s ) > max ( p, q ) ) return false;

if ( max ( r, s ) < min ( p, q ) ) return false;

return true;

}

**}**

**char lineAndLineIntersection2D ( myLine l, double &t, double &s ) {**

if ( b.cross2D ( l.b) == 0 ) {

if ( l.a.sub(a).cross2D(l.b) == 0 ) {

if ( overlapParallel ( l ) ) return 'o'; //overlaps

else return 'p'; //parallel

} else return 'd'; //disjoint and parallel

}

myVec w = a.sub ( l.a );

myVec p = l.b.perp2D(), z = b.perp2D();

t = -(w.dot(p))/p.dot(b); //for current line

s = w.dot(z)/z.dot(l.b); //for line l

return 'i';

**}**

**double lineAndLineDistance2D ( myLine l ) {**

double t, s; //First check if the intersect

char r = lineAndLineIntersection2D ( l, t, s );

if ( r == 'i' ) return 0; //Intersects. 0 distance.

//Parallel Lines

return lineToPointDistance ( l.a, t );

**}**

**double lineAndSegmentDistance2D ( myLine l ) {**

double t, s;

char r = lineAndLineIntersection2D ( l, t, s );

if ( r == 'i' && s + eps > 0 && s < 1 + eps ) {

return 0; //Valid intersection

}

double res = lineToPointDistance ( l.a, t );

res = min ( res, lineToPointDistance ( l.a.add(l.b), t ) );

return res;

**}**

**double segmentAndSegmentDistance2D ( myLine l ) {**

double t, s;

char r = lineAndLineIntersection2D ( l, t, s );

if ( r =='i' && t+eps > 0 && t < 1 + eps && s + eps > 0 && s < 1 + eps ) {

return 0; //Valid intersection

}

double res = segmentToPointDistance ( l.a, t );

res = min ( res, segmentToPointDistance ( l.a.add(l.b), t ) );

res = min ( res, l.segmentToPointDistance ( a, t ) );

res = min ( res, l.segmentToPointDistance ( a.add ( b ), t ) );

return res;

**}**

**myLine reflect ( myVec p, myVec norm ) {**

myVec ap = p.sub ( a ); //Starting to Point of Reflection

norm = norm.unitVec();

double d = fabs ( ap.dot ( norm ) );

myVec m = p.add ( norm.mul ( d ) );

myVec h = m.sub ( a ).mul ( 2 );

m = a.add ( h );

myLine ray = ray.lineFromPoints ( p, m );

return ray;

**}**

**};**

**struct myCir {**

myVec a;

double r;

**myVec atPos ( double t ) {**

myVec res;

res.val[0] = a.val[0] + r \* cos ( t );

res.val[1] = a.val[1] + r \* sin ( t );

return res;

**}**

**char circleAndLineIntersection2D ( myLine l, double &t1, double &t2 ) {**

double t3;

double d = l.lineToPointDistance ( a, t3 );

if ( d > r + eps ) return 'd';

if ( fabs ( d - r ) <= eps ) return 't';

myVec m = l.atPos ( t3 );

myVec am = m.sub ( a );

//Need to handle when line passes through center

double x = am.polarAngle2D();

double temp = d / r;

if ( temp > 1 ) temp = 1;

if ( temp < -1 ) temp = -1;

double theta = pi / 2 - asin ( temp ); //Using sin law find internal angle.

t1 = x + theta;

t2 = x - theta;

return 'i';

**}**

**char sphereAndLineIntersect ( myLine l, double &t1, double &t2 ) {**

double tp = 0;

double d = l.lineToPointDistance ( a, tp );

if ( d > r + eps ) return 'd';

if ( fabs ( d - r ) < eps ) {

t1 = tp;

return 't';

}

double chord = sqrt ( r \* r - d \* d );

t1 = tp - chord / l.b.length();

t2 = tp + chord / l.b.length();

return 'i';

**}**

**char circleAndCircleIntersection2D ( myCir c2, double &t1, double &t2 ) {**

myVec d = c2.a.sub ( a );

if ( d.length() > r + c2.r + eps ) return 'd'; //Case 1

if ( d.length() + c2.r + eps < r ) return 'd'; //Case 2

if ( a == c2.a && fabs ( r - c2.r ) <= eps ) {

if ( r == 0 ) {

t1 = 0;

return 't'; //Case 7

}

return 's'; //Case 6

}

if ( fabs ( d.length() - r - c2.r ) <= eps ||

fabs ( d.length() + c2.r - r ) <= eps ) {

t1 = d.polarAngle2D();

return 't'; //Case 3 and 4

}

double theta = cosineRule3Side ( r, d.length(), c2.r );

double m = d.polarAngle2D ();

t1 = m - theta;

t2 = m + theta;

return 'i'; //Case 5

**}**

**int circleToCircleTangentLine (myCir c2,myLine &l1,myLine &l2,myLine &l3,myLine &l4)** {

//First circle must be smaller or equal to second circle

if (r>c2.r + eps ) return c2.circleToCircleTangentLine ( \*this, l1, l2, l3, l4 );

myVec oo = c2.a.sub ( a );

double d = oo.length();

if ( fabs ( d ) < eps && fabs ( r - c2.r ) < eps ) //Infinite tangents

return -1;

if ( d + r + eps < c2.r ) //No tangents

return 0;

double base = oo.polarAngle2D();

if ( fabs ( d + r - c2.r ) < eps ) { //Contains Circle

l1 = l1.lineFromPoints ( atPos ( base + pi ), atPos ( base + pi ) );

return 1;

}

double ang = pi - acos ( (c2.r - r ) / d );

l1 = l1.lineFromPoints ( atPos ( base + ang ), c2.atPos ( base + ang ) );

l2 = l2.lineFromPoints ( atPos ( base - ang ), c2.atPos ( base - ang ) );

if ( d + eps < r + c2.r ) return 2; //Circle intersects

if ( fabs ( d - r - c2.r ) < eps ) { //Circle tangent

l3 = l3.lineFromPoints ( atPos ( base ), atPos ( base ) );

return 3;

}

//Disjoint Circle

ang = acos ( ( c2.r + r ) / d );

l3 = l3.lineFromPoints ( atPos ( base + ang ), c2.atPos ( base + ang + pi ) );

l4 = l4.lineFromPoints ( atPos ( base - ang ), c2.atPos ( base - ang + pi ) );

return 4;

**}**

**};** **//Complete Geometry Structure**

**bool collinear ( myVec a, myVec b, myVec c ) {**

myVec ab = b.sub(a), ac = c.sub(a);

double d = fabs ( ab.dot(ac) );

if ( fabs ( d - ab.length() \* ac.length() ) <= eps ) return true;

return false;

**}**

**bool collinear ( myVec a, myVec b, myVec c ) {**

myVec ab = b.sub(a), ac = c.sub(a);

double d = fabs ( ab.dot(ac) );

if ( fabs ( d - ab.length() \* ac.length() ) <= eps ) return true;

return false;

**}**

//Find if C is between A and B or B and A

**bool pointBetween ( pii a, pii b, pii c ) {**

if ( MIN(a.ff,b.ff) <= c.ff && c.ff <= MAX(a.ff,b.ff) && MIN(a.ss,b.ss) <= c.ss && c.ss <= MAX(a.ss,b.ss) ) return true;

else return false;

**}**

//Determine if (a,b) and (c,d) line intersects. All points are integer

**bool integerPointLineIntersect ( pii a, pii b, pii c, pii d ) {**

int s1 = triArea( a, b, c );

int s2 = triArea( a, b, d );

int s3 = triArea( c, d, a );

int s4 = triArea( c, d, b );

if ( s1 \* s2 > 0 || s3 \* s4 > 0 ) return false;

if ( s1 && s2 && s3 && s4 ) return true;

return ( between( a, b, c ) || between( a, b, d ) || between( c, d, a ) || between( c, d, b ) );

**}**

**Nafis Bhai Geometry Library**

using namespace std;

#define xx first

#define yy second

#define pb push\_back

#define mp make\_pair

#define LL long long

//#define inf INT\_MAX/3

#define mod 1000000007ll

#define PI 2.0\*acos(0.0)

#define linf (1ll<<60)-1

#define FOR(I,A,B) for(int I = (A); I < (B); ++I)

#define REP(I,N) FOR(I,0,N)

#define ALL(A) ((A).begin(), (A).end())

#define set0(ar) memset(ar,0,sizeof ar)

#define vsort(v) sort(v.begin(),v.end())

#define setinf(ar) memset(ar,126,sizeof ar)

//cout << fixed << setprecision(20) << p << endl;

template <class T> inline T bigmod(T p,T e,T M){

LL ret = 1;

for(; e > 0; e >>= 1){

if(e & 1) ret = (ret \* p) % M;

p = (p \* p) % M;

} return (T)ret;

}

template <class T> inline T gcd(T a,T b){if(b==0)return a;return gcd(b,a%b);}

template <class T> inline T modinverse(T a,T M){return bigmod(a,M-2,M);}

**const double eps=1e-8;**

**const double pi=acos(-1.0);**

**const double inf=1e20;**

**const int maxp=1111;**

**int dblcmp(double d){if (fabs(d)<eps)return 0;return d>eps?1:-1;}**

**inline double sqr(double x){return x\*x;}**

///\*

//point() - Empty constructor

//point(double x, double y) - constructor

//input() - double input

//output() - .2lf output

//operator == - compares x and y

//operator < - compares first by x, then by y

//len() - gives length from origin

//len2() - gives square of length from origin

//distance(point p) - gives distance from p

//add(point p) - returns new point after adding curresponging x and y

//sub(point p) - returns new point after subtracting curresponging x and y

//mul(double b) - returns new point after multiplieing x and y by b

//div(double b) - returns new point after divideing x and y by b

//dot(point p) - dot product

//det(point p) - cross product of 2d points

//rad(point a, point b) - Probably radius of circumcircle of the triangle

//trunc(double r) - return point that is truncated the distance from center to r

//rotleft() - returns 90 degree ccw rotated point

//rotright() - returns 90 degree cw rotated point

//rotate(point p, double angle) - returns point after rotateing the point centering at p by angle radian ccw

//\*/

**struct point**

**{**

double x,y;

**point() { }**

**point(double \_x,double \_y){ x = \_x; y = \_y; }**

**void input()** { int a,b;scanf("%d%d",&a,&b);x=a\*1.0,y=b\*1.0; }

**void output() { printf("%.2f %.2f\n",x,y); }**

**bool operator==(point a)const{**

return dblcmp(a.x - x) == 0 && dblcmp(a.y - y) == 0;

**}**

**bool operator<(point a)const{**

return dblcmp(a.x - x) == 0 ? dblcmp(y - a.y) < 0 : x < a.x;

**}**

**point operator-(point a)const{**

return point(x-a.x, y-a.y);

**}**

**double len() { return hypot(x, y); }**

**double len2() { return x \* x + y \* y; }**

**double distance(point p){return hypot(x - p.x, y - p.y); }**

**point add(point p) { return point(x + p.x, y + p.y); }**

**point sub(point p) { return point(x - p.x, y - p.y); }**

**point mul(double b) { return point(x \* b, y \* b); }**

**point div(double b) { return point(x / b, y / b); }**

**double dot(point p) { return x\*p.x+y\*p.y; }**

**double det(point p) { return x\*p.y-y\*p.x; }**

**double rad(point a,point b){**

point p=\*this;

return fabs(atan2(fabs(a.sub(p).det(b.sub(p))),a.sub(p).dot(b.sub(p))));

**}**

**point trunc(double r){**

double l=len();

if (!dblcmp(l))return \*this;

r/=l;

return point(x\*r,y\*r);

**}**

**point rotleft() { return point(-y,x); }**

**point rotright() { return point(y,-x); }**

**point rotate(point p,double angle){**

point v=this->sub(p);

double c=cos(angle),s=sin(angle);

return point(p.x+v.x\*c-v.y\*s,p.y+v.x\*s+v.y\*c);

**}**

};

///\*

//Stores two points

//

//line() - Empty constructor

//line(point a, point b) - line through a and b

//operator == - checks if two points are same

//line(point p, double angle) - one end p, another end at angle degree

//line(double a, double b, double c) - line of equation ax + by + c = 0

//input() - inputs a and b

//adjust() - orders in such a way that a < b

//length() - distance of ab

//angle() - returns 0 <= angle < 180

//relation() - 0 if collinear

// 1 if ccw

// 2 if cw

//pointonseg(point p) - returns 1 if point is on segment

//parallel(line v) - returns 1 if they are parallel

//segcrossseg(line v) - returns 0 if does not intersect

// returns 1 if non-standard intersection

// returns 2 if intersects

//segcrossseg\_inside(line v) - returns 1 if intersects strictly inside

// returns 0 if not

//linecrossseg(line v) - v is line

//linecrossline(line v) - 0 if parallel

// 1 if coincides

// 2 if intersects

//crosspoint(line v) - returns intersection point

//dispointtoline(point p) - distance from point p to the line

//dispointtoseg(point p) - distance from p to the segment

//lineprog(point p) - returns projected point p on ab line

//symmetrypoint(point p) - returns reflection point of p over ab

//\*/

**struct line**

**{**

point a,b;

**line() { }**

**line(point \_a,point \_b){ a=\_a; b=\_b; }**

**bool operator==(line v){ return (a==v.a)&&(b==v.b); }**

**line(point p,double angle){**

a=p;

if (dblcmp(angle-pi/2)==0){

b=a.add(point(0,1));

}else{

b=a.add(point(1,tan(angle)));

}

**}**

//ax+by+c=0

**line(double \_a,double \_b,double \_c){**

if (dblcmp(\_a)==0){

a=point(0,-\_c/\_b);

b=point(1,-\_c/\_b);

}else if (dblcmp(\_b)==0){

a=point(-\_c/\_a,0);

b=point(-\_c/\_a,1);

}else{

a=point(0,-\_c/\_b);

b=point(1,(-\_c-\_a)/\_b);

}

**}**

**void input() { a.input(); b.input(); }**

**void adjust() { if(b<a)swap(a,b); }**

**double length() { return a.distance(b); }**

**double angle(){**

double k=atan2(b.y-a.y,b.x-a.x);

if (dblcmp(k)<0)k+=pi;

if (dblcmp(k-pi)==0)k-=pi;

return k;

**}**

**int relation(point p){**

int c=dblcmp(p.sub(a).det(b.sub(a)));

if (c<0)return 1;

if (c>0)return 2;

return 3;

**}**

**bool pointonseg(point p){**

return dblcmp(p.sub(a).det(b.sub(a)))==0&&dblcmp(p.sub(a).dot(p.sub(b)))<=0;

**}**

**bool parallel(line v){**

return dblcmp(b.sub(a).det(v.b.sub(v.a)))==0;

**}**

**int segcrossseg(line v){**

int d1=dblcmp(b.sub(a).det(v.a.sub(a)));

int d2=dblcmp(b.sub(a).det(v.b.sub(a)));

int d3=dblcmp(v.b.sub(v.a).det(a.sub(v.a)));

int d4=dblcmp(v.b.sub(v.a).det(b.sub(v.a)));

if ((d1^d2)==-2&&(d3^d4)==-2)return 2;

return (d1==0&&dblcmp(v.a.sub(a).dot(v.a.sub(b)))<=0||

d2==0&&dblcmp(v.b.sub(a).dot(v.b.sub(b)))<=0||

d3==0&&dblcmp(a.sub(v.a).dot(a.sub(v.b)))<=0||

d4==0&&dblcmp(b.sub(v.a).dot(b.sub(v.b)))<=0);

**}**

**int segcrossseg\_inside(line v){**

if(v.pointonseg(a) || v.pointonseg(b) || pointonseg(v.a) || pointonseg(v.b)) return 0;

int d1=dblcmp(b.sub(a).det(v.a.sub(a)));

int d2=dblcmp(b.sub(a).det(v.b.sub(a)));

int d3=dblcmp(v.b.sub(v.a).det(a.sub(v.a)));

int d4=dblcmp(v.b.sub(v.a).det(b.sub(v.a)));

if ((d1^d2)==-2&&(d3^d4)==-2)return 1;

return (d1==0&&dblcmp(v.a.sub(a).dot(v.a.sub(b)))<=0||

d2==0&&dblcmp(v.b.sub(a).dot(v.b.sub(b)))<=0||

d3==0&&dblcmp(a.sub(v.a).dot(a.sub(v.b)))<=0||

d4==0&&dblcmp(b.sub(v.a).dot(b.sub(v.b)))<=0);

**}**

**int linecrossseg(line v){//\*this seg v line**

int d1=dblcmp(b.sub(a).det(v.a.sub(a)));

int d2=dblcmp(b.sub(a).det(v.b.sub(a)));

if ((d1^d2)==-2)return 2;

return (d1==0||d2==0);

**}**

**int linecrossline(line v){**

if ((\*this).parallel(v)){

return v.relation(a)==3;

}

return 2;

**}**

**point crosspoint(line v){**

double a1=v.b.sub(v.a).det(a.sub(v.a));

double a2=v.b.sub(v.a).det(b.sub(v.a));

return point((a.x\*a2-b.x\*a1)/(a2-a1),(a.y\*a2-b.y\*a1)/(a2-a1));

**}**

**double dispointtoline(point p){**

return fabs(p.sub(a).det(b.sub(a)))/length();

**}**

**double dispointtoseg(point p){**

if (dblcmp(p.sub(b).dot(a.sub(b)))<0||dblcmp(p.sub(a).dot(b.sub(a)))<0){

return min(p.distance(a),p.distance(b));

}

return dispointtoline(p);

**}**

**point lineprog(point p){**

return a.add(b.sub(a).mul(b.sub(a).dot(p.sub(a))/b.sub(a).len2()));

**}**

**point symmetrypoint(point p){**

point q=lineprog(p);

return point(2\*q.x-p.x,2\*q.y-p.y);

**}**

**};**

//a circle of point p and radius r

//

//circle() -empty constructor

//circle(point p,double r) -circle of point p and radius r

//circle(point a,point b,point c) -circumcircle of triangle of abc

//circle(point a,point b,point c,bool t)-incircle of triangle of abc, bool t is nothing

//input() -takes input of a circle

//output() -outputs a circle

//operator== -checks for equality

//operator< -comparison opertaor

//area() -area of the circle

//circumference() -circumference of the circle

//relation(point p) -0 outside

// 1 on circumference

// 2 inside circle

//relationseg(line v) -0 outside

// 1 on circumference

// 2 inside circle

//relationline(line v) -0 outside

// 1 on circumference

// 2 inside circle

//getcircle(point a,point b,double r,circle&c1,circle&c2)

// -returns two circle c1,c2 through points a,b of radius r

// returns 0 for nor circle

//getcircle(line u,point q,double r1,circle &c1,circle &c2)

// -returns two circle c1,c2 which is tangent to line u, goes through

// point q and has radius r1

// returns 0 for no circle ,1 if c1=c2 ,2 if c1!=c2

//getcircle(line u,line v,double r1,circle &c1,circle &c2,circle &c3,circle &c4)

// -returns 4 circles which is tangent to line u,v has radius r1.

//getcircle(circle cx,circle cy,double r1,circle&c1,circle&c2)

// -not sure

//pointcrossline(line v,point &p1,point &p2)

// -not sure

//relationcircle(circle v) -1 for

// 2

// 3

// 4

// 5

//pointcrosscircle(circle v,point &p1,point &p2)

// -not sure what it does

//tangentline(point q,line &u,line &v) -not sure what it does

//areacircle(circle v) -intersection area of circle v

//areatriangle(point a,point b) -intersection area of circle and triangle of point a,b,p

**struct circle**

**{**

point p;

double r;

**circle() { }**

**circle(point \_p,double \_r): p(\_p),r(\_r){ };**

**circle(double x,double y,double \_r): p(point(x,y)),r(\_r){};**

**circle(point a,point b,point c){**

p=line(a.add(b).div(2),a.add(b).div(2).add(b.sub(a).rotleft())).crosspoint(line(c.add(b).div(2),c.add(b).div(2).add(b.sub(c).rotleft())));

r=p.distance(a);

**}**

**circle(point a,point b,point c,bool t){**

line u,v;

double m=atan2(b.y-a.y,b.x-a.x),n=atan2(c.y-a.y,c.x-a.x);

u.a=a;

u.b=u.a.add(point(cos((n+m)/2),sin((n+m)/2)));

v.a=b;

m=atan2(a.y-b.y,a.x-b.x),n=atan2(c.y-b.y,c.x-b.x);

v.b=v.a.add(point(cos((n+m)/2),sin((n+m)/2)));

p=u.crosspoint(v);

r=line(a,b).dispointtoseg(p);

**}**

**void input() { p.input();scanf("%lf",&r); }**

**void output() { printf("%.2lf %.2lf %.2lf\n",p.x,p.y,r); }**

**bool operator==(circle v){**

return ((p==v.p)&&dblcmp(r-v.r)==0);

**}**

**bool operator<(circle v)const{**

return ((p<v.p)||(p==v.p)&&dblcmp(r-v.r)<0);

**}**

**double area() { return pi\*sqr(r); }**

**double circumference(){ return 2\*pi\*r; }**

**int relation(point b){**

double dst=b.distance(p);

if (dblcmp(dst-r)<0)return 2;

if (dblcmp(dst-r)==0)return 1;

return 0;

**}**

**int relationseg(line v){**

double dst=v.dispointtoseg(p);

if (dblcmp(dst-r)<0)return 2;

if (dblcmp(dst-r)==0)return 1;

return 0;

**}**

**int relationline(line v){**

double dst=v.dispointtoline(p);

if (dblcmp(dst-r)<0)return 2;

if (dblcmp(dst-r)==0)return 1;

return 0;

**}**

**int getcircle(point a,point b,double r,circle&c1,circle&c2){**

circle x(a,r),y(b,r);

int t=x.pointcrosscircle(y,c1.p,c2.p);

if (!t)return 0;

c1.r=c2.r=r;

return t;

**}**

**int getcircle(line u,point q,double r1,circle &c1,circle &c2){**

double dis=u.dispointtoline(q);

if (dblcmp(dis-r1\*2)>0)return 0;

if (dblcmp(dis)==0){

c1.p=q.add(u.b.sub(u.a).rotleft().trunc(r1));

c2.p=q.add(u.b.sub(u.a).rotright().trunc(r1));

c1.r=c2.r=r1;

return 2;

}

line u1=line(u.a.add(u.b.sub(u.a).rotleft().trunc(r1)),u.b.add(u.b.sub(u.a).rotleft().trunc(r1)));

line u2=line(u.a.add(u.b.sub(u.a).rotright().trunc(r1)),u.b.add(u.b.sub(u.a).rotright().trunc(r1)));

circle cc=circle(q,r1);

point p1,p2;

if (!cc.pointcrossline(u1,p1,p2))cc.pointcrossline(u2,p1,p2);

c1=circle(p1,r1);

if (p1==p2) { c2=c1;return 1; }

c2=circle(p2,r1);

return 2;

**}**

**int getcircle(line u,line v,double r1,circle &c1,circle &c2,circle &c3,circle &c4){**

if (u.parallel(v))return 0;

line u1=line(u.a.add(u.b.sub(u.a).rotleft().trunc(r1)),u.b.add(u.b.sub(u.a).rotleft().trunc(r1)));

line u2=line(u.a.add(u.b.sub(u.a).rotright().trunc(r1)),u.b.add(u.b.sub(u.a).rotright().trunc(r1)));

line v1=line(v.a.add(v.b.sub(v.a).rotleft().trunc(r1)),v.b.add(v.b.sub(v.a).rotleft().trunc(r1)));

line v2=line(v.a.add(v.b.sub(v.a).rotright().trunc(r1)),v.b.add(v.b.sub(v.a).rotright().trunc(r1)));

c1.r=c2.r=c3.r=c4.r=r1;

c1.p=u1.crosspoint(v1);

c2.p=u1.crosspoint(v2);

c3.p=u2.crosspoint(v1);

c4.p=u2.crosspoint(v2);

return 4;

**}**

**int getcircle(circle cx,circle cy,double r1,circle&c1,circle&c2){**

circle x(cx.p,r1+cx.r),y(cy.p,r1+cy.r);

int t=x.pointcrosscircle(y,c1.p,c2.p);

if (!t)return 0;

c1.r=c2.r=r1;

return t;

**}**

**int pointcrossline(line v,point &p1,point &p2){**

if (!(\*this).relationline(v))return 0;

point a=v.lineprog(p);

double d=v.dispointtoline(p);

d=sqrt(r\*r-d\*d);

if (dblcmp(d)==0){ p1=a; p2=a; return 1; }

p1=a.sub(v.b.sub(v.a).trunc(d));

p2=a.add(v.b.sub(v.a).trunc(d));

return 2;

**}**

**int relationcircle(circle v){**

double d=p.distance(v.p);

if (dblcmp(d-r-v.r)>0)return 5;

if (dblcmp(d-r-v.r)==0)return 4;

double l=fabs(r-v.r);

if (dblcmp(d-r-v.r)<0&&dblcmp(d-l)>0)return 3;

if (dblcmp(d-l)==0)return 2;

if (dblcmp(d-l)<0)return 1;

**}**

**int pointcrosscircle(circle v,point &p1,point &p2){**

int rel=relationcircle(v);

if (rel==1||rel==5)return 0;

double d=p.distance(v.p);

double l=(d+(sqr(r)-sqr(v.r))/d)/2;

double h=sqrt(sqr(r)-sqr(l));

p1=p.add(v.p.sub(p).trunc(l).add(v.p.sub(p).rotleft().trunc(h)));

p2=p.add(v.p.sub(p).trunc(l).add(v.p.sub(p).rotright().trunc(h)));

if (rel==2||rel==4)return 1;

return 2;

**}**

**int tangentline(point q,line &u,line &v){**

int x=relation(q);

if (x==2)return 0;

if (x==1){

u=line(q,q.add(q.sub(p).rotleft()));

v=u; return 1;

}

double d=p.distance(q);

double l=sqr(r)/d;

double h=sqrt(sqr(r)-sqr(l));

u=line(q,p.add(q.sub(p).trunc(l).add(q.sub(p).rotleft().trunc(h))));

v=line(q,p.add(q.sub(p).trunc(l).add(q.sub(p).rotright().trunc(h))));

return 2;

**}**

**double areacircle(circle v){**

int rel=relationcircle(v);

if (rel>=4)return 0.0;

if (rel<=2)return min(area(),v.area());

double d=p.distance(v.p);

double hf=(r+v.r+d)/2.0;

double ss=2\*sqrt(hf\*(hf-r)\*(hf-v.r)\*(hf-d));

double a1=acos((r\*r+d\*d-v.r\*v.r)/(2.0\*r\*d));

a1=a1\*r\*r;

double a2=acos((v.r\*v.r+d\*d-r\*r)/(2.0\*v.r\*d));

a2=a2\*v.r\*v.r;

return a1+a2-ss;

**}**

**double areatriangle(point a,point b){**

if (dblcmp(p.sub(a).det(p.sub(b))==0))return 0.0;

point q[5];

int len=0;

q[len++]=a;

line l(a,b);

point p1,p2;

if (pointcrossline(l,q[1],q[2])==2){

if (dblcmp(a.sub(q[1]).dot(b.sub(q[1])))<0)q[len++]=q[1];

if (dblcmp(a.sub(q[2]).dot(b.sub(q[2])))<0)q[len++]=q[2];

}

q[len++]=b;

if (len==4&&(dblcmp(q[0].sub(q[1]).dot(q[2].sub(q[1])))>0))swap(q[1],q[2]);

double res=0;

int i;

for (i=0;i<len-1;i++){

if (relation(q[i])==0||relation(q[i+1])==0){

double arg=p.rad(q[i],q[i+1]);

res+=r\*r\*arg/2.0;

}

else res+=fabs(q[i].sub(p).det(q[i+1].sub(p))/2.0);

}

return res;

}

**};**

///\*

//n, p, line l for each side

//

//input(n) - inputs n size polygon

//add(point p) - adds a point at end of the list

//getline() - populates line array

//cmp - comparison in convex\_hull order

//norm() - sorting in convex\_hull order

//getconvex(polygon &convex) - returns convex hull in convex (monotone chain)

//isconvex() - checks if convex

//relationpoint(point q) - returns 3 if q is a vertex

// 2 if on a side

// 1 if inside

// 0 if outside

//relationline(line u) - returns 1 if there is some intersection

// 0 if no intersection

// 2 if intersect at corner

//convexcut(line u,polygon &po) - left side of u in po

//getcircumference() - returns side length

//getarea() - returns area

//getdir() - returns 0 for cw, 1 for ccw

//getbarycentre() - returns barycenter / cg

//areaintersection(polygon po) - not implemented

//areaunion(polygon po) - not implemented

//areacircle(circle c) - intersection area of circle and polygon

//relationcircle(circle c) - returns 0 if outside circle

// 1 if tangent

// 2 if inside

//mincircle() - returns minimum enclosing circle

//circlecover() - i think there is mistake. it tries to find minimum enclosing circle

//pointinpolygon(point q) - -1 if not on polygon, non negative number.. side index

//inside\_polygon(point q, int on\_edge=1)

// - returns on\_edge if on edge, otherwise 0 for outside 1 for inside

//isdiagonal(int a, int b) - checks if p[a], p[b] is diagonal or not. returns 0/1

//\*/

**struct polygon**

**{**

int n;

point p[maxp];

line l[maxp];

**void input(int \_n){**

n=\_n;

for (int i=0;i<n;i++) p[i].input();

**}**

**void add(point q) { p[n++]=q; }**

**void getline(){**

for (int i=0;i<n;i++)

l[i]=line(p[i],p[(i+1)%n]);

**}**

**struct cmp{**

point p;

cmp(const point &p0){p=p0;}

bool operator()(const point &aa,const point &bb){

point a=aa,b=bb;

int d=dblcmp(a.sub(p).det(b.sub(p)));

if (d==0)

return dblcmp(a.distance(p)-b.distance(p))<0;

return d>0;

}

**};**

**void norm(){**

point mi=p[0];

for (int i=1;i<n;i++)mi=min(mi,p[i]);

sort(p,p+n,cmp(mi));

**}**

**void getconvex(polygon &convex){**

int i;

sort(p,p+n);

convex.n=n;

for (i=0;i<min(n,2);i++) convex.p[i]=p[i];

if (n<=2)return;

int &top=convex.n;

top=1;

for (i=2;i<n;i++){

while (top&&convex.p[top].sub(p[i]).det(convex.p[top-1].sub(p[i]))<=0)

top--;

convex.p[++top]=p[i];

}

int temp=top;

convex.p[++top]=p[n-2];

for (i=n-3;i>=0;i--){

while (top!=temp&&convex.p[top].sub(p[i]).det(convex.p[top-1].sub(p[i]))<=0)

top--;

convex.p[++top]=p[i];

}

**}**

**bool isconvex(){**

bool s[3];

memset(s,0,sizeof(s));

int i,j,k;

for (i=0;i<n;i++){

j=(i+1)%n;

k=(j+1)%n;

s[dblcmp(p[j].sub(p[i]).det(p[k].sub(p[i])))+1]=1;

if (s[0]&&s[2])return 0;

}

return 1;

**}**

**int relationpoint(point q){**

int i,j;

for (i=0;i<n;i++){

if (p[i]==q)return 3;

}

getline();

for (i=0;i<n;i++){

if (l[i].pointonseg(q))return 2;

}

int cnt=0;

for (i=0;i<n;i++){

j=(i+1)%n;

int k=dblcmp(q.sub(p[j]).det(p[i].sub(p[j])));

int u=dblcmp(p[i].y-q.y);

int v=dblcmp(p[j].y-q.y);

if (k>0&&u<0&&v>=0)cnt++;

if (k<0&&v<0&&u>=0)cnt--;

}

return cnt!=0;

**}**

**int relationline(line u){**

int i,k=0;

getline();

for (i=0;i<n;i++){

if (l[i].segcrossseg(u)==2)return 1;

if (l[i].segcrossseg(u)==1)k=1;

}

if (!k)return 0;

vector<point>vp;

for (i=0;i<n;i++){

if (l[i].segcrossseg(u)){

if (l[i].parallel(u)){

vp.pb(u.a);

vp.pb(u.b);

vp.pb(l[i].a);

vp.pb(l[i].b);

continue;

}

vp.pb(l[i].crosspoint(u));

}

}sort(vp.begin(),vp.end());

int sz=vp.size();

for (i=0;i<sz-1;i++){

point mid=vp[i].add(vp[i+1]).div(2);

if (relationpoint(mid)==1)return 1;

}

return 2;

**}**

**void convexcut(line u,polygon &po){**

int i;

int &top=po.n;

top=0;

for (i=0;i<n;i++){

int d1=dblcmp(p[i].sub(u.a).det(u.b.sub(u.a)));

int d2=dblcmp(p[(i+1)%n].sub(u.a).det(u.b.sub(u.a)));

if (d1>=0)po.p[top++]=p[i];

if (d1\*d2<0)po.p[top++]=u.crosspoint(line(p[i],p[(i+1)%n]));

}

**}**

**double getcircumference(){**

double sum=0;

int i;

for (i=0;i<n;i++)

sum+=p[i].distance(p[(i+1)%n]);

return sum;

**}**

**double getarea(){**

double sum=0;

int i;

for (i=0;i<n;i++)

sum+=p[i].det(p[(i+1)%n]);

return fabs(sum)/2;

**}**

**bool getdir(){**

double sum=0;

int i;

for (i=0;i<n;i++)

sum+=p[i].det(p[(i+1)%n]);

if (dblcmp(sum)>0)return 1;

return 0;

**}**

**point getbarycentre(){**

point ret(0,0);

double area=0;

int i;

for (i=1;i<n-1;i++){

double tmp=p[i].sub(p[0]).det(p[i+1].sub(p[0]));

if (dblcmp(tmp)==0)continue;

area+=tmp;

ret.x+=(p[0].x+p[i].x+p[i+1].x)/3\*tmp;

ret.y+=(p[0].y+p[i].y+p[i+1].y)/3\*tmp;

}

if (dblcmp(area))ret=ret.div(area);

return ret;

**}**

**double areaintersection(polygon po){ }**

**double areaunion(polygon po){**

return getarea()+po.getarea()-areaintersection(po);

**}**

**double areacircle(circle c){**

int i,j,k,l,m;

double ans=0;

for (i=0;i<n;i++){

int j=(i+1)%n;

if (dblcmp(p[j].sub(c.p).det(p[i].sub(c.p)))>=0)

ans+=c.areatriangle(p[i],p[j]);

else ans-=c.areatriangle(p[i],p[j]);

}

return fabs(ans);

**}**

**int relationcircle(circle c){**

getline();

int i,x=2;

if (relationpoint(c.p)!=1)return 0;

for (i=0;i<n;i++){

if (c.relationseg(l[i])==2)return 0;

if (c.relationseg(l[i])==1)x=1;

}

return x;

**}**

**void find(int st,point tri[],circle &c){**

if (!st) c=circle(point(0,0),-2);

if (st==1) c=circle(tri[0],0);

if (st==2) c=circle(tri[0].add(tri[1]).div(2),tri[0].distance(tri[1])/2.0);

if (st==3) c=circle(tri[0],tri[1],tri[2]);

**}**

**void solve(int cur,int st,point tri[],circle &c){**

find(st,tri,c);

if (st==3)return;

int i;

for (i=0;i<cur;i++){

if (dblcmp(p[i].distance(c.p)-c.r)>0){

tri[st]=p[i];

solve(i,st+1,tri,c);

}

}

**}**

**circle mincircle(){**

random\_shuffle(p,p+n);

point tri[4];

circle c;

solve(n,0,tri,c);

return c;

**}**

**int circlecover(double r){**

int ans=0,i,j;

vector<pair<double,int> >v;

for (i=0;i<n;i++){

v.clear();

for (j=0;j<n;j++)if (i!=j){

point q=p[i].sub(p[j]);

double d=q.len();

if (dblcmp(d-2\*r)<=0){

double arg=atan2(q.y,q.x);

if (dblcmp(arg)<0)arg+=2\*pi;

double t=acos(d/(2\*r));

v.push\_back(make\_pair(arg-t+2\*pi,-1));

v.push\_back(make\_pair(arg+t+2\*pi,1));

}

}

sort(v.begin(),v.end());

int cur=0;

for (j=0;j<v.size();j++){

if (v[j].second==-1)++cur;

else --cur;

ans=max(ans,cur);

}

}

return ans+1;

**}**

**};**

**Int main(){**

point a();

line b;

circle c;

polygon q;

int x,y,r,u,v,cs=1;

int t;

cin>>t;

while(t--)

{

cin>>x>>y>>r;

c=circle(point(x\*1.0,y\*1.0),r\*1.0);

cin>>u>>v>>x>>y;

q.n=4;

q.p[0]=point(u\*1.0,v\*1.0);

q.p[1]=point(x\*1.0,v\*1.0);

q.p[2]=point(x\*1.0,y\*1.0);

q.p[3]=point(u\*1.0,y\*1.0);

printf("Case %d: %.10lf\n",cs++,q.areacircle(c));

}

}

**Point In Polygon**

**int crossproduct(pair<int,int> a , pair<int,int> b , pair<int,int>p ){**

return ((a.x-p.x)\*(b.y-p.y)-(a.y-p.y)\*(b.x-p.x)) ;

}

**bool isInsidePolygon( pair<int,int> P[] , pair<int,int> sample , int n ){**

int cnt = 0 ;

bool f = 0 ;

for( int i = 0 , j = n-1 ; i < n ; j = i++ ){

int Xmin = min( P[i].x,P[j].x ) ;

int Xmax = max( P[i].x,P[j].x ) ;

int Ymin = min( P[i].y,P[j].y ) ;

int Ymax = max( P[i].y,P[j].y ) ;

if( Xmin <= sample.x && sample.x <= Xmax && Ymin <= sample.y && sample.y <= Ymax){

if( crossproduct( P[i] , P[j] , sample ) == 0 )return true ;

}

if( crossproduct( P[i] , P[j] , sample ) < 0 ){

swap( P[i],P[j] ) ;

f = 1 ;

}

if( P[i].y >= sample.y && P[j].y < sample.y )cnt++ ;

if( f == 1 ){

f = 0 ;

swap( P[i],P[j] ) ;

}

}

return ( cnt & 1 ) ;

}

**Tan Of Line**

**pair<int,int> TanOfLine( pair<int,int> a , pair<int,int> b )**

{

const int INF = 1<<30 ;

pair<int,int> M ;

if( a.y == b.y ) {

M.x = INF ;

M.y = INF ;

} else {

M.x = a.x-b.x ;

M.y = a.y-b.y ;

int tmp = \_\_gcd( (M.x),(M.y) ) ;

M.x = M.x/tmp ;

M.y = M.y/tmp ;

}

return M ;

}

**/\***

**closestPair(Point \*X, Point \*Y, int n);**

**X contains the points sorted by x co-ordinate,**

**Y contains the points sorted by y co-ordinate,**

**One additional item in Point structure is needed, the original index.**

**\*/**

typedef long long i64;

typedef struct { int x, y, i; } Point;

int flag[MAX];

**inline i64 sq(const i64 &x) {**

return x\*x;

**}**

**inline i64 sqdist(const Point &a, const Point &b) {**

return sq(a.x-b.x) + sq(a.y-b.y);

**}**

**inline i64 closestPair(Point \*X, Point \*Y, int n) {**

if(n == 1) return INF;

if(n == 2) return sqdist(X[0], X[1]);

int i, j, k, n1, n2, ns, m = n >> 1;

Point Xm = X[m-1], \*XL, \*XR, \*YL, \*YR, \*YS;

i64 lt, rt, dd, tmp;

XL = new Point[m], YL = new Point[m];

XR = new Point[m+1], YR = new Point[m+1];

YS = new Point[n];

for(i = 0; i < m; i++) XL[i] = X[i], flag[X[i].i] = 0;

for(; i < n; i++) XR[i - m] = X[i], flag[X[i].i] = 1;

for(i = n2 = n1 = 0; i < n; i++) {

if(!flag[Y[i].i]) YL[n1++] = Y[i];

else YR[n2++] = Y[i];

}

lt = closestPair(XL, YL, n1);

rt = closestPair(XR, YR, n2);

dd = min(lt, rt);

for(i = ns = 0; i < n; i++)

if(sq(Y[i].x - Xm.x) < dd)

YS[ns++] = Y[i];

for(j = 0; j < ns; j++)

for(k = j + 1; k < ns && sq(YS[k].y - YS[j].y) < dd; k++)

dd = min(dd, sqdist(YS[j], YS[k]));

delete[] XL; delete[] XR;

delete[] YL; delete[] YR;

delete[] YS;

return dd;

**}**

**/\***

**ConvexHull : Graham's Scan O(n lg n), integer implementation**

**P[]: holds all the points, C[]: holds points on the hull**

**np: number of points in P[], nc: number of points in C[]**

**to handle duplicate, call makeUnique() before calling convexHull()**

**call convexHull() if you have np >= 3**

**to remove co-linear points on hull, call compress() after convexHull()**

**\*/**

point P[MAX], C[MAX], P0;

**inline int triArea2(const point &a, const point &b, const point &c) {**

return (a.x\*(b.y-c.y) + b.x\*(c.y-a.y) + c.x\*(a.y-b.y));

**}**

**inline int sqDist(const point &a, const point &b) {**

return ((a.x-b.x)\*(a.x-b.x) + (a.y-b.y)\*(a.y-b.y));

**}**

**inline bool comp(const point &a, const point &b) {**

int d = triArea2(P0, a, b);

if(d < 0) return false;

if(!d && sqDist(P0, a) > sqDist(P0, b)) return false;

return true;

**}**

**inline bool normal(const point &a, const point &b) {**

return ((a.x==b.x) ? a.y < b.y : a.x < b.x);

**}**

**inline bool issame(const point &a, const point &b) {**

return (a.x == b.x && a.y == b.y);

**}**

**inline void makeUnique(int &np) {**

sort(&P[0], &P[np], normal);

np = unique(&P[0], &P[np], issame) - P;

**}**

**void convexHull(int &np, int &nc) {**

int i, j, pos = 0;

for(i = 1; i < np; i++)

if(P[i].y<P[pos].y || (P[i].y==P[pos].y && P[i].x<P[pos].x))

pos = i;

swap(P[0], P[pos]);

P0 = P[0];

sort(&P[1], &P[np], comp);

for(i = 0; i < 3; i++) C[i] = P[i];

for(i = j = 3; i < np; i++) {

while(triArea2(C[j-2], C[j-1], P[i]) < 0) j--;

C[j++] = P[i];

}

nc = j;

**}**

**void compress(int &nc) {**

int i, j, d;

C[nc] = C[0];

for(i=j=1; i < nc; i++) {

d = triArea2(C[j-1], C[i], C[i+1]);

if(d || (!d && issame(C[j-1], C[i+1]))) C[j++] = C[i];

}

nc = j;

**}**

**/\***

**C[] array of points of convex polygon in ccw order,**

**nc number of points in C, p target points.**

**returns true if p is inside C (including edge) or false otherwise.**

**complexity O(lg n)**

**\*/**

inline bool inConvexPoly(point \*C, int nc, const point &p) {

int st = 1, en = nc - 1, mid;

while(en - st > 1) {

mid = (st + en)>>1;

if(triArea2(C[0], C[mid], p) < 0) en = mid;

else st = mid;

}

if(triArea2(C[0], C[st], p) < 0) return false;

if(triArea2(C[st], C[en], p) < 0) return false;

if(triArea2(C[en], C[0], p) < 0) return false;

return true;

}

**/\***

**P[] holds the points, must be either in cw or ccw**

**function returns double of the area.**

**\*/**

**inline int dArea(int np) {**

int area = 0;

for(int i = 0; i < np; i++) {

area += p[i].x\*p[i+1].y - p[i].y\*p[i+1].x;

}

return abs(area);

**}**

**/\***

**This code assumes the circle center and radius to be integer.**

**Change this when necessary.**

**\*/**

**inline double commonArea(const Circle &a, const Circle &b) {**

int dsq = sqDist(a.c, b.c);

double d = sqrt((double)dsq);

if(sq(a.r + b.r) <= dsq) return 0;

if(a.r >= b.r && sq(a.r-b.r) >= dsq) return pi \* b.r \* b.r;

if(a.r <= b.r && sq(b.r-a.r) >= dsq) return pi \* a.r \* a.r;

double angleA = 2.0 \* acos((a.r \* a.r + dsq - b.r \* b.r) / (2.0 \* a.r \* d));

double angleB = 2.0 \* acos((b.r \* b.r + dsq - a.r \* a.r) / (2.0 \* b.r \* d));

return 0.5 \* (a.r \* a.r \* (angleA - sin(angleA)) + b.r \* b.r \* (angleB - sin(angleB)));

**}**

**/\***

**Segment intersection in 2D integer space.**

**P1, p2 makes first segment, p3, p4 makes the second segment**

**\*/**

**inline bool intersect(const Point &p1, const Point &p2, const Point &p3, const Point &p4) {**

i64 d1, d2, d3, d4;

d1 = direction(p3, p4, p1);

d2 = direction(p3, p4, p2);

d3 = direction(p1, p2, p3);

d4 = direction(p1, p2, p4);

if(((d1 < 0 && d2 > 0) || (d1 > 0 && d2 < 0)) && ((d3 < 0 && d4 > 0) || (d3 > 0 && d4 < 0))) return true;

if(!d3 && onsegment(p1, p2, p3)) return true;

if(!d4 && onsegment(p1, p2, p4)) return true;

if(!d1 && onsegment(p3, p4, p1)) return true;

if(!d2 && onsegment(p3, p4, p2)) return true;

return false;

**}**

**/\***

**Some tetrahedron formulas**

**\*/**

**inline double volume(double u, double v, double w, double U, double V, double W) {**

double u1,v1,w1;

u1 = v \* v + w \* w - U \* U;

v1 = w \* w + u \* u - V \* V;

w1 = u \* u + v \* v - W \* W;

return sqrt(4.0\*u\*u\*v\*v\*w\*w - u\*u\*u1\*u1 - v\*v\*v1\*v1 - w\*w\*w1\*w1 + u1\*v1\*w1) / 12.0;

**}**

**inline double surface(double a, double b, double c) {**

return sqrt((a + b + c) \* (-a + b + c) \* (a - b + c) \* (a + b - c)) / 4.0;

**}**

**inline double insphere(double WX, double WY, double WZ, double XY, double XZ, double YZ) {**

double sur, rad;

sur = surface(WX, WY, XY) + surface(WX, XZ, WZ) + surface(WY, YZ, WZ) + surface(XY, XZ, YZ);

rad = volume(WX, WY, WZ, YZ, XZ, XY) \* 3.0 / sur;

return rad;

**}**

**/\***

**1. Assign hull.n**

**2. Take input in hull.point**

**3. Call convexSort()**

**4. Call findHull()**

**5. Convex Hull is ready in hull.convex with hull.m points in it.**

**\*/**

**pll g;**

**vlong triArea ( pll a, pll b, pll c ) {**

vlong area = a.ff \* b.ss + b.ff \* c.ss + c.ff \* a.ss;

area -= a.ff \* c.ss + b.ff \* a.ss + c.ff \* b.ss;

return area;

**}**

**vlong sqDist ( pll a, pll b ) {**

return ( SQ(a.ff-b.ff) + SQ(a.ss-b.ss ) );

**}**

**bool convexCompare ( const pll &a, const pll &b ) {**

vlong area = triArea ( g, a, b );

if ( area > 0 ) return true;

else if ( area == 0 && sqDist ( g, a ) < sqDist ( g, b ) ) return true;

else return false;

**}**

**struct ConvexHull {**

int n, m;

pll point[PPP], convex[PPP];

**void convexSort() {**

g = point[0];

FOR(i,0,n-1) {

if ( point[i].ff < g.ff ) g = point[i];

else if ( point[i].ff == g.ff && point[i].ss < g.ss ) g = point[i];

}

sort ( point, point + n, convexCompare );

**}**

**void findHull() {**

if ( n == 1 ) {

convex[0] = convex[1] = point[0];

m = 1;

return;

}

convex[0] = point[n-1]; convex[1] = point[0]; convex[2] = point[1];

int cur = 3;

for ( int i = 2; i < n; i++ ) {

vlong area = triArea ( convex[cur-2], convex[cur-1], point[i] );

if ( area > 0 ) {

convex[cur] = point[i];

cur++;

}

else if ( area == 0 ) { ///Take action depending on what is required

/\*Left Vertical Line gets omitted. Manually handle it\*/

/\*convex[cur] = point[i];

cur++;\*/

///If extra point needs to be removed

convex[cur-1] = point[i];

}

else {

cur--;

i--;

}

}

m = cur - 1;

}

**}hull;**

**Monotone Chain**

**#include<bits/stdc++.h>**

**#define pi acos(-1.0)**

**using namespace std ;**

**struct Point {**

long long x, y;

int idx ;

bool operator <(const Point &p) const {

return x < p.x || (x == p.x && y < p.y);

}

**};**

**int n ;double d ;vector<Point> arr ;**

**long long cross(const Point &O, const Point &A, const Point &B){**

return (A.x - O.x) \* (B.y - O.y) - (A.y - O.y) \* (B.x - O.x);

**}**

**vector<Point> convex\_hull(vector<Point> P)**

**{**

int n = P.size(), k = 0;

vector<Point> H(2\*n);

sort(P.begin(), P.end());

for (int i = 0; i < n; ++i) {

while (k >= 2 && cross(H[k-2], H[k-1], P[i]) < 0) k--;

H[k++] = P[i];

}

for (int i = n-2, t = k+1; i >= 0; i--) {

while (k >= t && cross(H[k-2], H[k-1], P[i]) < 0) k--;

H[k++] = P[i];

}

H.resize(k);

return H;

**}**

**INHABITANT**

**pair<long long,long long> P[100000+5] ;**

int n ;

**long long isLeft( pair<long long,long long> a , pair<long long,long long> b , pair<long long,long long> c ) {**

//does point b is left or right to ac segment

return (a.x\*(b.y-c.y) + b.x\*(c.y-a.y) + c.x\*(a.y-b.y));

**}**

**bool check( long long x, long long y ) {**

int lo = 1 , hi = n-1 ; ;

int mid ;

pair<long long,long long>sample = make\_pair(x,y) ;

while( hi-lo >= 2 ) {

mid = (lo+hi)>>1 ;

if( isLeft( P[0], P[mid] , sample ) >= 0 )lo = mid ;

else hi = mid ;

}

long long a = isLeft(P[0], P[lo], sample) ;

if(isLeft(P[0], P[lo], sample) < 0) return 0;

if(isLeft(P[lo], P[hi], sample) < 0) return 0;

if(isLeft(P[hi], P[0], sample) < 0) return 0;

return 1 ;

**}**

**Angle Rotation**

**pair<double , double> Rotate\_Point( pair<double,double> P ,pair<double,double> Ref ,double dist, double cosA ,double sinA ){**

//if only theta comes then sinA= sin(theta) , conA = cos( theta ) ;

double a = sqrt( (P.first-Ref.first)\*(P.first-Ref.first) + (P.second-Ref.second)\*(P.second-Ref.second) ) ;

double x1 = (dist\*(Ref.first-P.first))/a ;

double y1 = (dist\*(Ref.second-P.second))/a ;

pair<double,double> ret ;

ret.first = x1\*cosA-y1\*sinA+P.first ;

ret.second = x1\*sinA+y1\*cosA+P.second ;

return ret ;

**}**

|  |  |
| --- | --- |
| **double d = b\*b-a\*c ;**  **double n = a\*c\*(a+2\*b+c) ;**  **double ans ;**  **if( d == 0 ) ans = -1 ;**  **else ans = n / d ;** |  |

**// Assume that a class is already given for the object:**

**// Point with coordinates {float x, y;}**

**//===================================================================**

**// isLeft(): test if a point is Left|On|Right of an infinite line.**

**// Input: three points P0, P1, and P2**

**// Return: >0 for P2 left of the line through P0 and P1**

**// =0 for P2 on the line**

**// <0 for P2 right of the line**

**// See: Algorithm 1 on Area of Triangles**

**inline float**

**isLeft( Point P0, Point P1, Point P2 )**

**{**return (P1.x - P0.x)\*(P2.y - P0.y) - (P2.x - P0.x)\*(P1.y - P0.y);**}**

**// simpleHull\_2D(): Melkman's 2D simple polyline O(n) convex hull algorithm**

**// Input: P[] = array of 2D vertex points for a simple polyline**

**// n = the number of points in V[]**

**// Output: H[] = output convex hull array of vertices (max is n)**

**// Return: h = the number of points in H[]**

**Int simpleHull\_2D( Point\* P, int n, Point\* H )**

**{**

// initialize a deque D[] from bottom to top so that the

// 1st three vertices of P[] are a ccw triangle

Point\* D = new Point[2\*n+1];

int bot = n-2, top = bot+3; // initial bottom and top deque indices

D[bot] = D[top] = P[2]; // 3rd vertex is at both bot and top

if (isLeft(P[0], P[1], P[2]) > 0) {

D[bot+1] = P[0];

D[bot+2] = P[1]; // ccw vertices are: 2,0,1,2

}

else {

D[bot+1] = P[1];

D[bot+2] = P[0]; // ccw vertices are: 2,1,0,2

}

// compute the hull on the deque D[]

for (int i=3; i < n; i++) { // process the rest of vertices

// test if next vertex is inside the deque hull

if ((isLeft(D[bot], D[bot+1], P[i]) > 0) &&

(isLeft(D[top-1], D[top], P[i]) > 0) )

continue; // skip an interior vertex

// incrementally add an exterior vertex to the deque hull

// get the rightmost tangent at the deque bot

while (isLeft(D[bot], D[bot+1], P[i]) <= 0)

++bot; // remove bot of deque

D[--bot] = P[i]; // insert P[i] at bot of deque

// get the leftmost tangent at the deque top

while (isLeft(D[top-1], D[top], P[i]) <= 0)

--top; // pop top of deque

D[++top] = P[i]; // push P[i] onto top of deque

}

// transcribe deque D[] to the output hull array H[]

int h; // hull vertex counter

for (h=0; h <= (top-bot); h++)

H[h] = D[bot + h];

delete D;

return h-1;

**}**

**Beauty Contest**

|  |  |  |
| --- | --- | --- |
| **Time Limit:** 3000MS |  | **Memory Limit:** 65536K |
| **Total Submissions:** 23523 |  | **Accepted:** 7182 |

**Description**

Bessie, Farmer John's prize cow, has just won first place in a bovine beauty contest, earning the title 'Miss Cow World'. As a result, Bessie will make a tour of N (2 <= N <= 50,000) farms around the world in order to spread goodwill between farmers and their cows. For simplicity, the world will be represented as a two-dimensional plane, where each farm is located at a pair of integer coordinates (x,y), each having a value in the range -10,000 ... 10,000. No two farms share the same pair of coordinates.   
  
Even though Bessie travels directly in a straight line between pairs of farms, the distance between some farms can be quite large, so she wants to bring a suitcase full of hay with her so she has enough food to eat on each leg of her journey. Since Bessie refills her suitcase at every farm she visits, she wants to determine the maximum possible distance she might need to travel so she knows the size of suitcase she must bring.Help Bessie by computing the maximum distance among all pairs of farms.

**Input**

\* Line 1: A single integer, N   
  
\* Lines 2..N+1: Two space-separated integers x and y specifying coordinate of each farm

**Output**

\* Line 1: A single integer that is the squared distance between the pair of farms that are farthest apart from each other.

**Sample Input**

4

0 0

0 1

1 1

1 0

**Sample Output**

2

typedef vector <int> vi;

typedef pair <int,int> pii;

#define FOR(i, a, b) for (i = (a); i <= (b); i++)

#define FORD(i, a, b) for (i = (a); i >= (b); i--)

#define REP(i, a) for (i = 0; i < (a); i++)

#define REPD(i, a) for (i = (a) - 1; i >= 0; i--)

#define ALL(v) (v).begin(), (v).end()

#define SET(a, x) memset((a), (x), sizeof(a))

#define SZ(a) ((int)(a).size())

#define CL(a) ((a).clear())

#define SORT(x) sort(ALL(x))

#define mp make\_pair

#define pb push\_back

#define filer() freopen("in.txt","r",stdin)

#define MAX(a,b) ((a) > (b) ? (a) : (b))

#define MIN(a,b) ((a) < (b) ? (a) : (b))

#define EPS 1e-10

#define S(x) ((x)\*(x))

#define INF 999999999

using namespace std;

double PI=acos(-1.0);

struct Vector{

double x,y;

void scan(){scanf("%lf %lf",&x,&y);}

void print(){printf("(%lf,%lf)\n",x,y);}

Vector(double x=0,double y=0){

this->x=x;

this->y=y;

}

double mag2(){

return S(x)+S(y);

}

double mag(){return sqrt(mag2());}

};

double dotp(Vector a, Vector b){

return a.x\*b.x + a.y\*b.y ;

}

double crossp(Vector a, Vector b){

return a.x\*b.y - a.y\*b.x;

}

Vector operator+(Vector a,Vector b){

return Vector(a.x+b.x, a.y+b.y); }

Vector operator-(Vector a,Vector b){

return Vector(a.x-b.x, a.y-b.y); }

Vector operator\*(Vector a, double b){

return Vector(a.x\*b, a.y\*b); }

double angle(Vector a, Vector b){

double v= dotp(a,b) / (a.mag()\*b.mag()) ;

return acos(v);

}

int Turn(Vector &V0,Vector &V1,Vector &V2)

{

double v=crossp((V1-V0),(V2-V0));

if(fabs(v)<EPS)return 0;

if(v>0)return 1;

return -1;

}

Vector V[50007];

int stk[50007];

bool cmp(Vector V1,Vector V2)

{

int t=Turn(V[0],V1,V2);

if(!t)

{

return (V1-V[0]).mag2()<(V2-V[0]).mag2();

}

return t>0;

}

int hull(int N)

{

int i,id=0;

int top=0;

if(N<=2)

{

stk[top++]=0;

if(N==2)stk[top++]=1;

return top;

}

for(i=1;i<N;i++)

{

if(V[i].y==V[id].y)

{

if(V[i].x<V[id].x)id=i;

}

if(V[i].y<V[id].y)id=i;

}

swap(V[0],V[id]);

sort(V+1,V+N,cmp);

stk[top++]=0;

stk[top++]=1;

for(i=2;i<N;i++)

{

while(Turn(V[stk[top-2]],V[stk[top-1]],V[i])<1&&top>1)top--;

stk[top++]=i;

}

return top;

}

inline Vector get(Vector &u,double a)

{

Vector v=Vector(-u.y,u.x);

return Vector(u\*cos(a)+v\*sin(a));

}

int main()

{

// filer();

int i,N;

while(scanf("%d",&N)==1)

{

REP(i,N)V[i].scan();

int top=hull(N);

int lo=0,hi=0;

Vector ref1,ref2;

REP(i,top)

{

if(V[stk[lo]].y>V[stk[i]].y)lo=i;

if(V[stk[hi]].y<V[stk[i]].y)hi=i;

}

int ans=-INF;

ref1=Vector(1,0);

ref2=Vector(-1,0);

int tlo=lo,thi=hi;

int nlo,nhi;

double ang1,ang2;

double tang=0,ang;

while(tang<PI)

{

// cout<<hi<<" "<<lo<<endl;

ans=MAX(ans,(V[stk[hi]]-V[stk[lo]]).mag2());

nlo=(lo+1)%top;

nhi=(hi+1)%top;

ang1=angle(V[stk[nlo]]-V[stk[lo]],ref1);

ang2=angle(V[stk[nhi]]-V[stk[hi]],ref2);

ang=MIN(ang1,ang2);

if(ang1<ang2)

{

ref1=get(ref1,ang1);

ref2=get(ref2,ang1);

lo=nlo;

tang+=ang1;

}

else

{

ref1=get(ref1,ang2);

ref2=get(ref2,ang2);

hi=nhi;

tang+=ang2;

}

}

printf("%d\n",ans);

}

return 0;

}

**FFT**

#include<bits/stdc++.h>

using namespace std;

const double PI=acos(-1.0);

typedef complex<double> base;

const int MAXN=800005;

base wlen\_pw[MAXN];

typedef complex<double> base;

**int rev (int num, int lg\_n) {**

int res = 0;

for (int i=0; i<lg\_n; ++i)

if (num & (1<<i))

res |= 1<<(lg\_n-1-i);

return res;

**}**

**void fft (vector<base> & a, bool invert) {**

int n = (int) a.size();

int lg\_n = 0;

while ((1 << lg\_n) < n) ++lg\_n;

for (int i=0; i<n; ++i)

if (i < rev(i,lg\_n))

swap (a[i], a[rev(i,lg\_n)]);

for (int len=2; len<=n; len<<=1) {

double ang = 2\*PI/len \* (invert ? -1 : 1);

base wlen (cos(ang), sin(ang));

for (int i=0; i<n; i+=len) {

base w (1);

for (int j=0; j<len/2; ++j) {

base u = a[i+j], v = a[i+j+len/2] \* w;

a[i+j] = u + v;

a[i+j+len/2] = u - v;

w \*= wlen;

}

}

}

if (invert)

for (int i=0; i<n; ++i)

a[i] /= n;

**}**

**void multiply (const vector<int> & a, const vector<int> & b, vector<int> & res) {**

vector<base> fa (a.begin(), a.end()), fb (b.begin(), b.end());

size\_t n = 1;

while (n < max (a.size(), b.size())) n <<= 1;

n <<= 1;

fa.resize (n), fb.resize (n);

fft (fa, false), fft (fb, false);

for (size\_t i=0; i<n; ++i)

fa[i] \*= fb[i];

fft (fa, true);

res.resize (n);

for (size\_t i=0; i<n; ++i)

res[i] = int (fa[i].real() + 0.5);

**}**

**Aho Korasik**

#include<bits/stdc++.h>

using namespace std ;

char T[1000000+1] ;

char keyword[500] ;

char tmp[1000000+1] ;

int n ;

int freq[1000000+1] ;

int ans[1000000] ;

**struct Trie {**

int level,\_next[26] ;

vector<int>patIdx ;

int parent ;

**Trie() {**

level = 0 ;

memset( \_next , 0 , sizeof \_next ) ;

patIdx.clear() ;

parent = 0 ;

**}**

**};**

const int MAXC = 500\*500+5 ;

Trie Tree[MAXC] ;

int treeIdx ;

**void InsertTrie(char keyword[],int idx) {**

int root = 0 ;

for( int i = 0 ; keyword[i] ; i++ ) {

char ch = keyword[i]-'a' ;

if( Tree[root].\_next[ch] == 0 ) {

Tree[root].\_next[ch] = treeIdx++ ;

}

Tree[Tree[root].\_next[ch]].level = Tree[root].level+1 ;

root = Tree[root].\_next[ch] ;

}

Tree[root].patIdx.push\_back(idx);

**}**

**void print(){**

for( int i = 0 ;i < 9 ; i++ )cout << Tree[i].parent << " " ;

cout << endl ;

**}**

**int FindParent( int src,int ch ) {**

int par = Tree[src].parent ;

while( par > 0 && Tree[par].\_next[ch] == 0 ) {

par = Tree[par].parent ;

}

return par ;

**}**

**void bfsOnTrie() {**

queue<int>Q ;

for( int i =0 ; i < 26 ; i++ ) {

if( Tree[0].\_next[i] != 0 ) {

Q.push( Tree[0].\_next[i] ) ;

}

}

while( !Q.empty() ) {

int src = Q.front() ;

Q.pop();

for( int i = 0 ; i < 26 ; i++ ) {

if( Tree[src].\_next[i] != 0 ) {

int dest = Tree[src].\_next[i] ;

int par = FindParent(src,i) ;

Tree[dest].parent = Tree[par].\_next[i] ;

//print();

Q.push(dest) ;

}

}

}

**}**

**bool comp( pair<int,int> a , pair<int,int> b ){**

return a.first > b.first ;

**}**

**void query() {**

int root = 0 ;

memset( freq , 0 , sizeof freq ) ;

for( int i = 0 ; T[i] ; i++ ) {

int ch = T[i] - 'a' , par;

if( Tree[root].\_next[ch] == 0 ) {

int par = FindParent(root,ch) ;

root = Tree[par].\_next[ch] ;

} else {

root = Tree[root].\_next[ch] ;

}

freq[root]++ ;

}

vector<pair<int,int> > tmp ;

for( int i = 0 ; i < treeIdx ; i++ ){

tmp.push\_back( make\_pair(Tree[i].level,i) ) ;

}

sort( tmp.begin() , tmp.end() , comp );

for( int i = 0 ; i < treeIdx ; i++ ){

freq[ Tree[tmp[i].second].parent ]+=freq[tmp[i].second] ;

}

for( int i = 1 ; i < treeIdx ; i++ ){

for( int j = 0 ; j < Tree[i].patIdx.size() ; j++ ){

ans[Tree[i].patIdx[j]] = freq[i] ;

}

}

**}**

**int main() {**

int cases,caseno=1 ;

scanf("%d",&cases ) ;

while( cases -- ) {

scanf("%d%s",&n,T) ;

treeIdx = 1 ;

for( int i = 0 ; i < n ; i++ ) {

scanf("%s",keyword) ;

InsertTrie(keyword,i) ;

}

bfsOnTrie() ;

query();

cout << "Case " << caseno++ << ":\n";

for(int i = 0 ; i < n ; i++ ){

cout << ans[i] << "\n" ;

}

for( int i = 0 ; i < MAXC ; i++ ) Tree[i] = Trie() ;

}

return 0 ;

**}**

**/\***

**1**

**4**

**aishers**

**he**

**she**

**hers**

**his**

**\*/**

**KMP**

#include<bits/stdc++.h>

using namespace std;

**//KMP p=A^m ;**

**int main() {**

ios\_base::sync\_with\_stdio(0);

cin.tie(nullptr);

string s;

while(cin>>s && s!=".") {

if(s.size()==1) {

cout<<"1\n";

continue;

}

int jump[s.size()];

jump[0]=0;

for(int i=1,q=0; i<(int)s.size(); i++) {

while(q>0 && s[i]!=s[q])q=jump[q-1];

if(s[i]==s[q])q++;

jump[i]=q;

}

int x=s.size()-jump[s.size()-1];

if(s.size()%x==0) {

cout<<s.size()/x<<"\n";

} else cout<<"1\n";

}

return 0;

**}**

**//KMP**

int\_next[1000000+1] ;

**void failure( string T ) {**

int \_left = 0 , \_right ;

\_next[0] = 0 ;

for( \_right = 1 ; \_right < T.size() ; \_right ++ ) {

while( \_left > 0 && T[\_left] != T[\_right] ) {

\_left = \_next[\_left-1] ;

}

if( T[\_left] == T[\_right] )\_left++ ;

\_next[\_right] = \_left ;

}

**}**

**int KMP(string T,string P){**

int \_left = 0 , \_right , cnt = 0 ;

for( \_right = 0 ; \_right < T.size() ; \_right ++ ){

while( \_left > 0 && P[\_left] != T[\_right] ) \_left = \_next[\_left-1] ;

if( P[\_left] == T[\_right] )\_left ++ ;

if( \_left == P.size() ) {

cnt ++ ;

\_left = \_next[\_left-1] ;

}

}

return cnt ;

**}**

**//Z**

int M[1000000] ;

int z[1000000] ;

string s ;

void calc() {

memset(z,0,sizeof z) ;

int L = 0, R = 0;

int n = s.size() ;

for (int i = 1; i < n ; i++) {

if (i > R) {

L = R = i;

while (R < n && s[R-L] == s[R]) R++;

z[i] = R-L;

R--;

} else {

int k = i-L;

if (z[k] < R-i+1) z[i] = z[k];

else {

L = i;

while (R < n && s[R-L] == s[R]) R++;

z[i] = R-L;

R--;

}

}

}

}

**// Text Repetition**

#include<bits/stdc++.h>

using namespace std ;

int \_next[1000000+2];

**void failure( string s ){**

\_next[0] = 0 ;

for( int i = 1,q=0 ; s[i] ; i++ ){

while( q > 0 && s[q] != s[i] )q = \_next[q-1] ;

if( s[q] == s[i] ) q++ ;

\_next[i] = q ;

}

**}**

**int main() {**

ios\_base::sync\_with\_stdio(0) ;

cin.tie(0);

int n ;

string s ;

while(cin >> n >> s ) {

if( n == -1 && s =="\*")break ;

if( s.size() > n ){

cout << 0 << '\n' ;

continue ;

}else if( s.size()== n ){

cout << 1 << '\n' ;

continue ;

}

failure(s);

int k = \_next[s.size()-1] ;

cout << 1+(n-s.size())/(s.size()-k) << "\n" ;

}

return 0 ;

**}**

**/\***

**14 abcab**

**1000 abcde**

**1000000000 z**

**1 zzzzz**

**-1 \***

**output**

**4**

**200**

**1000000000**

**0**

**14 “abcab” == “abcabcabcabcab”**

**ans is 4**

**\*/**

**/\***

**Manacher algorithm implementation.**

**Application, largest palindromic substring, largest palindromic suffix**

**\*/**

**int lengths[MAX<<1];**

**int manacher(char \*buff, int len) {**

int i, k, pallen, found, d, j, s, e;

k = pallen = 0;

for(i = 0; i < len; ) {

if(i > pallen && buff[i-pallen-1] == buff[i]) {

pallen += 2, i++;

continue;

}

lengths[k++] = pallen;

s = k - 2, e = s - pallen, found = 0;

for(j = s; j > e; j--) {

d = j - e - 1;

if(lengths[j] == d) {

pallen = d;

found = 1;

break;

}

lengths[k++] = (d < lengths[j]? d : lengths[j]);

}

if(!found) { pallen = 1; i++; }

}

lengths[k++] = pallen;

return lengths[k-1];

**}**

**BIT**

**/\***

**update and query function for 2D bit.**

**MAX is the maximum possible value.**

**bit[][] holds the 2D binary indexed tree**

**\*/**

**void update(int x, int y, int v) {**

int y1;

while(x <= MAX) {

y1 = y;

while(y1 <= MAX) {

bit[x][y1] += v;

y1 += (y1 & -y1);

}

x += (x & -x);

}

**}**

**int readsum(int x, int y) {**

int v = 0, y1;

while(x > 0) {

y1 = y;

while(y1 > 0) {

v += bit[x][y1];

y1 -= (y1 & -y1);

}

x -= (x & -x);

}

return v;

**}**

**(III) Range Update, Range Query:**

I didn't even know it exists until I read some post in TopCoder forums (the post was made by: AnilKishore).

I will just re-state his explanation as it was quite clear itself. All we want here is to support range updates, and do range queries. As from the previous type, if we try to store range updates, the BIT structure effectively captures updates for single positions, instead of range/cumulative sums. However, we can do some tweaking to work around this problem.

Let's just consider only one update: Add v to [a, b] while the rest elements of the array is 0.

Now, consider sum(0, x) for all possible x, again three situation can arise:

1. 0 ≤ x < a : which results in 0

2. a ≤ x ≤ b : we get v \* (x - (a-1))

3. b < x < n : we get v \* (b - (a-1))

This suggests that, if we can find v\*x for any index x, then we can get the sum(0, x) by subtracting T from it, where:

1. 0 ≤ x < : Sum should be 0, thus, T = 0

2. a ≤ x ≤ : Sum should be v\*x-v\*(a-1), thus, T = v\*(a-1)

3. b < x < n : Sum should be 0, thus, T = -v\*b + v\*(a-1)

As, we can see, knowing T solves our problem, we can use another BIT to store this additive amount from which we can get:

0 for x < a, v\*(a-1) for x in [a..b], -v\*b+v(a-1) for x > b.

Now we have two BITs.

To add v in range [a, b]: Update(a, v), Update(b+1, -v) in the first BIT and Update(a, v\*(a-1)) and Update(b+1, -v\*b) on the second BIT.

To get sum in range [0, x]: you simply do Query\_BIT1(x)\*x - Query\_BIT2(x);

Now you know how to find range sum for [a, b]. Just find sum(b) - sum(a-1) using the formula stated above.

Pretty impressive this one! SPOJ - HORRIBLE can be solved in this approach. And thanks to Iqram Mahmud for this nice problem.

(IV) 2D BIT

How to write Update and Query methods for 2D BIT is well described in the TopCoder tutorial I've mentioned above. The only thing to notice here is how the queries and updates work. 2D BIT is basically a BIT where each element is another BIT. Adding v on (x, y) means it's effect will be found throughout the rectangle [(x, y), (W, H)], and query for (x, y) gives you the result of the rectangle [(0, 0), (x, y)], assuming the total rectangle is [(0, 0), (W, H)]. SO when you query and update on this BIT, you have to be careful about how many times you are subtracting a rectangle and adding it. Simple set union formula works here. So if you want to get the result of a specific rectangle [(x1, y1), (x2, y2)], the following steps are necessary to do so:

V = Query(x2, y2)

V -= Query(x2, y1-1)

V -= Query(x1-1, y2)

V += Query(x1-1, y1-1)

This problem is an example: SPOJ - MATSUM

This page is likely to be updated if I find and manage to solve new types of BIT problems. Please let me know if there are mistakes. This post is inspired by some random posts lying around in TopCoder forums, I just wanted to put them in one place for future reference.

**long long tree[100000+2] ;**

**void update( int idx, int val ) {**

while( idx<= 1001 ) {

tree[idx] += val ;

idx += ( idx & -idx );

}

**}**

**long long query( int idx ) {**

long long sum = 0 ;

while( idx > 0 ) {

sum += tree[idx] ;

idx -= ( idx & -idx ) ;

}

return sum ;

**}**

**void range\_update( int a, int b, int v ) {**

update(a,v) ;

update(b+1,-v) ;

**}**

**long long point\_query(int x) {**

return query(x) ;

**}**

**LCA**

//LCA using sparse table

//Complexity: O(NlgN,lgN)

#define EX 100002

int L[EX];

int P[EX][22];

int T[EX];

vector<int>g[EX];

**void dfs(int from,int u,int dep)**

**{**

T[u]=from;

L[u]=dep;

for(int i=0;i<(int)g[u].size();i++)

{

int v=g[u][i];

if(v==from) continue;

dfs(u,v,dep+1);

}

**}**

**int lca\_query(int N, int p, int q)**

**{**

int tmp, log, i;

if (L[p] < L[q])

tmp = p, p = q, q = tmp;

log=1;

while(1) {

int next=log+1;

if((1<<next)>L[p])break;

log++;

}

for (i = log; i >= 0; i--)

if (L[p] - (1 << i) >= L[q])

p = P[p][i];

if (p == q)

return p;

for (i = log; i >= 0; i--)

if (P[p][i] != -1 && P[p][i] != P[q][i])

p = P[p][i], q = P[q][i];

return T[p];

**}**

**void lca\_init(int N)**

**{**

memset (P,-1,sizeof(P));

int i, j;

for (i = 0; i < N; i++)

P[i][0] = T[i];

for (j = 1; 1 << j < N; j++)

for (i = 0; i < N; i++)

if (P[i][j - 1] != -1)

P[i][j] = P[P[i][j - 1]][j - 1];

**}**

**int main(void) {**

g[0].pb(1);

g[0].pb(2);

g[2].pb(3);

g[2].pb(4);

dfs(0, 0, 0);

lca\_init(5);

printf( "%d\n", lca\_query(5,3,4) );

return 0;

**}**

**HLD**

**#include<bits/stdc++.h>**

**using namespace std;**

**inline int RI() {**

int ret = 0, flag = 1,ip = getchar\_unlocked();

for(; ip < 48 || ip > 57; ip = getchar\_unlocked()) {

if(ip == 45) {

flag = -1;

ip = getchar\_unlocked();

break;

}

}

for(; ip > 47 && ip < 58; ip = getchar\_unlocked())

ret = ret \* 10 + ip - 48 ;

return flag \* ret;

**}**

Int n, u, v, a[30004], child[30004], tree[30004], level[30004], pr[30004], head[30004], cnext[30004], chainid[30004], chainpos[30004], pos, nchain;

vector<int>g[30004];

**void update(int ii,int vv) {**

while(ii<=n) {

tree[ii]+=vv;

ii+=((ii)&(-ii));

}

return;

**}**

**int read(int ii) {**

int ret=0;

while(ii>0) {

ret+=tree[ii];

ii-=((ii)&(-ii));

}

return ret;

**}**

**void dfs(int s,int p) {**

child[s]=1;

for(int i=0; i<(int)g[s].size(); i++) {

if(g[s][i]!=p) {

level[g[s][i]]=level[s]+1;

pr[g[s][i]]=s;

dfs(g[s][i],s);

child[s]+=child[g[s][i]];

}

}

return;

**}**

**void hld(int s,int p) {**

if(head[nchain]==-1) {

head[nchain]=s;

}

chainid[s]=nchain;

chainpos[s]=pos;

pos++;

update(chainpos[s],a[s]);

int heavyid,mx=-1;

for(int i=0; i<(int)g[s].size(); i++) {

if(g[s][i]!=p) {

if(mx<child[g[s][i]]) {

mx=child[g[s][i]];

heavyid=g[s][i];

}

}

}

if(mx!=-1) {

hld(heavyid,s);

}

for(int i=0; i<(int)g[s].size(); i++) {

if(g[s][i]!=p && g[s][i]!=heavyid) {

nchain++;

hld(g[s][i],s);

}

}

return;

**}**

**inline int lca(int x,int y) {**

while(true) {

int xst=head[chainid[x]],yst=head[chainid[y]];

if(chainid[x]==chainid[y])return (level[x]<level[y])?x:y;

if(level[xst]<level[yst])y=pr[yst];

else x=pr[xst];

}

return 0;

**}**

**int hldquery(int x,int y) {**

int ret=0;

while(true) {

if(chainid[x]==chainid[y]) {

ret+=(read(chainpos[x])-read(chainpos[y]-1));

break;

}

ret+=read(chainpos[x])-read(chainpos[head[chainid[x]]]-1);

x=pr[head[chainid[x]]];

}

return ret;

**}**

**int main() {**

int t=RI();

for(int z=1; z<=t; z++) {

for(int i=0; i<n; i++) {

g[i].clear();

}

memset(head,-1,sizeof(head));

memset(pr,-1,sizeof(pr));

memset(tree,0,sizeof(tree));

nchain=1;

pos=1;

n=RI();

for(int i=0; i<n; i++) {

a[i]=RI();

}

for(int i=1; i<n; i++) {

u=RI(),v=RI();

g[u].push\_back(v);

g[v].push\_back(u);

}

level[0]=0;

pr[0]=-1;

dfs(0,-1);

hld(0,-1);

int q=RI(),tt,x,y;

cout<<"Case "<<z<<":\n";

while(q--) {

tt=RI(),x=RI(),y=RI();

if(tt==0) {

int \_lca=lca(x,y);

cout<<hldquery(x,\_lca)+hldquery(y,\_lca)-a[\_lca]<<"\n";

}else{

update(chainpos[x],-a[x]);

update(chainpos[x],y);

a[x]=y;

}

}

}

return 0;

**}**

**Tarjan’s Offline LCA**

**/\***

**Tarjan's offline LCA algorithm. For each pair of node's in P {u, v, qid},**

**it finds the LCA of the nodes in the rooted tree G (no edge to back to the parent.**

**The array ans holds the result for queries in orders defined by qid.**

**\*/**

**void lca(int u) {**

int v, i, sz;

make\_set(u);

ancestor[find\_set(u)] = u;

sz = G[u].size();

for(i = 0; i < sz; i++) {

v = G[u][i];

lca(v);

union\_set(u, v);

ancestor[find\_set(u)] = u;

}

color[u] = 1;

sz = P[u].size();

for(i = 0; i < sz; i++) {

v = P[u][i].first;

if(color[v]) ans[P[u][i].second] = ancestor[find\_set(v)];

}

**}**

**Order Static set**

#include<bits/stdc++.h>

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

#include <ext/pb\_ds/detail/standard\_policies.hpp>

using namespace \_\_gnu\_pbds;

using namespace std;

template<typename T>

using orderset =

tree<T, null\_type, less<T>, rb\_tree\_tag,

tree\_order\_statistics\_node\_update>;

**int main() {**

int cases = readInt() ;

orderset<int> mp;

while( cases -- ){

char ch ;

scanf("%c",&ch) ;

int n = readInt() ;

if( ch == 'I' ){

mp.insert( n ) ;

}else if( ch == 'D' ){

mp.erase(n) ;

}else if( ch == 'K' ){

if( n > mp.size() ){

cout << "invalid\n" ;

}else{

cout << \*mp.find\_by\_order(n-1) << "\n" ;

}

}else if( ch == 'C' ){

cout << mp.order\_of\_key(n) << "\n" ;

}

}

return 0 ;

**}**

**Trie**

struct trie{

int \_next[2],branch;

trie():branch(0){memset(\_next,-1,sizeof(\_next));}

}tree[200005\*40];

int tii=0;

**void \_insert(int x){**

int cur=0;

tree[cur].branch++;

for(int i=30;i>=0;i--){

int qw=((x>>i)&1);

if(tree[cur].\_next[qw]==-1){

tii++;

tree[cur].\_next[qw]=tii;

}

cur=tree[cur].\_next[qw];

tree[cur].branch++;

}

return;

**}**

**void \_delete(int x,int ii,int cur){**

tree[cur].branch--;

if(ii<0)return;

int qw=(x>>ii)&1;

\_delete(x,ii-1,tree[cur].\_next[qw]);

return;

**}**

**int \_find(int x){**

int ret=0;

int cur=0;

if(tree[cur].branch==0)return x;

for(int i=30;i>=0;i--){

int qw=(x>>i)&1;

if(tree[cur].\_next[1-qw]!=-1 && tree[tree[cur].\_next[1-qw]].branch>0){

ret=ret\*2+(1-qw);

cur=tree[cur].\_next[1-qw];

}else{

ret=ret\*2+qw;

cur=tree[cur].\_next[qw];

}

}return max(x,x^ret);

**}**

**struct trie {**

int next[2] ;

trie() {

next[0] = next[1] = -1 ;

}

**} tree[50002\*32] ;**

int arr[50002] , Xor[50002] , idx , temp ;

**void init(){**

for( int i = 0; i <= idx ; i++ ){

tree[i] = trie() ;

}

idx = 1 ;

**}**

**void Insert( int n ){**

int root = 0 ;

for( int i = 31 ; i >= 0 ; i-- ){

bool bit = (n&(1<<i)) ;

if( tree[root].next[bit] == -1 ){

tree[root].next[bit]=idx++ ;

}

root = tree[root].next[bit] ;

}

return ;

**}**

**int Query( int n , bool f ){**

int root = 0 , ret = 0 ;

for( int i = 31 ; i >= 0 ; i-- ){

bool bit = (n&(1<<i)) ;

if( !f ){

if( tree[root].next[1-bit] != -1 )bit = 1-bit ;

}else {

if( tree[root].next[bit] == -1 )bit = 1-bit ;

}

root = tree[root].next[bit] ;

ret = (ret\*2) + (int)bit;

}

return ret^n ;

**}**

**Disjoint set**

**struct DisjointSet {**

int \*root, \*rank, n;

**DisjointSet(int sz) {**

root = new int[sz+1];

rank = new int[sz+1];

n = sz;

**}**

**~DisjointSet() {**

delete[] root;

delete[] rank;

**}**

**void init() {**

for(int i = 1; i <= n; i++) {

root[i] = i;

rank[i] = 0;

}

**}**

**int find(int u) {**

if(u != root[u]) root[u] = find(root[u]);

return root[u];

**}**

**void merge(int u, int v) {**

int pu = find(u);

int pv = find(v);

if(rank[pu] > rank[pv]) root[pv] = pu;

else root[pu] = pv;

if(rank[pu]==rank[pv]) rank[pv]++;

**}**

**};**

**Centroid Decomposition**

Ciel the Commander

Now Fox Ciel becomes a commander of Tree Land. Tree Land, like its name said, has *n* cities connected by *n* - 1 undirected roads, and for any two cities there always exists a path between them.

Fox Ciel needs to assign an officer to each city. Each officer has a rank — a letter from 'A' to 'Z'. So there will be 26 different ranks, and 'A' is the topmost, so 'Z' is the bottommost.

There are enough officers of each rank. But there is a special rule must obey: if *x* and *y* are two distinct cities and their officers have the same rank, then on the simple path between *x* and *y* there must be a city *z* that has an officer with higher rank. The rule guarantee that a communications between same rank officers will be monitored by higher rank officer.

Help Ciel to make a valid plan, and if it's impossible, output "Impossible!".

**Input**

The first line contains an integer *n* (2 ≤ *n* ≤ 105) — the number of cities in Tree Land.

Each of the following *n* - 1 lines contains two integers *a* and *b* (1 ≤ *a*, *b* ≤ *n*, *a* ≠ *b*) — they mean that there will be an undirected road between *a* and *b*. Consider all the cities are numbered from 1 to *n*.

It guaranteed that the given graph will be a tree.

**Output**

If there is a valid plane, output *n* space-separated characters in a line — *i*-th character is the rank of officer in the city with number *i*.

Otherwise output "Impossible!".

**Examples**

**input**

4  
1 2  
1 3  
1 4

**output**

A B B B

**input**

10  
1 2  
2 3  
3 4  
4 5  
5 6  
6 7  
7 8  
8 9  
9 10

**output**

D C B A D C B D C D

**Note**

In the first example, for any two officers of rank 'B', an officer with rank 'A' will be on the path between them. So it is a valid solution.

vector<int> adj[100007];

int Size[100007];

int Rank[100007];

bool vis[100007];

int totSize;

void Dfs(int u,int p){

Size[u] = 1;

for (int i=0;i<adj[u].size();i++){

int v = adj[u][i];

if (v == p) continue;

if (vis[v]) continue;

Dfs(v,u);

Size[u] += Size[v];

}

}

int Centroid(int u,int p){

for (int i=0;i<adj[u].size();i++){

int v = adj[u][i];

if (v == p) continue;

if (vis[v]) continue;

if (Size[v] >= totSize/2)

return Centroid(v,u);

}

return u;

}

void Decompose(int u,int l,int p){

Dfs(u,-1);

totSize = Size[u];

u = Centroid(u,-1);

vis[u] = 1;

Rank[u] = 'A' + l;

for (int i=0;i<adj[u].size();i++){

int v = adj[u][i];

if (v == p) continue;

if (vis[v]) continue;

Decompose(v,l+1,u);

}

}

int main (){

int n;

scanf ("%d",&n);

for (int i=1;i<n;i++){

int u,v;

scanf ("%d%d",&u,&v);

adj[u].push\_back(v);

adj[v].push\_back(u);

}

Decompose(1,0,-1);

for (int i=1;i<=n;i++) printf("%c ",Rank[i]);

return 0;

}

**Matrix Expo**

**#include<bits/stdc++.h>**

**using namespace std ;**

**class matrix {**

public:

vector<vector<int> >arr;

**matrix() {}**

**matrix(int N) {**

for( int i = 0 ; i < N ; i ++ ) {

vector<int>y;

for( int j = 0 ; j < N ; j++ ) {

y.push\_back(0);

}

arr.push\_back(y);

}

**}**

**matrix operator \*(const matrix &in) {**

matrix ret ;

int N=this->arr.size();

ret=matrix(N);

for( int i = 0 ; i < N ; i++ ) {

for( int j = 0 ; j < N ; j++ )

for( int k = 0 ; k < N ; k++ ) {

ret.arr[i][j]+=(arr[i][k])\*(in.arr[k][j]) ;

ret.arr[i][j]%=10 ;

}

}

return ret ;

**}**

**matrix operator ^( long long int POW) {**

matrix ret;

int N=this->arr.size();

ret=matrix(N);

for( int i = 0 ; i < N ; i++ ) {

ret.arr[i][i] = 1 ;

}

matrix ME = \*this ;

while( POW ) {

if( POW&1 ) {

ret = ret \* ME ;

**}**

ME = ME \* ME ;

POW >>= 1 ;

}

return ret ;

**}**

**matrix operator +(const matrix &x) {**

matrix ret;

int N=this->arr.size();

ret=matrix(N);

for(int i=0; i<N; i++) {

for(int j=0; j<N; j++) {

ret.arr[i][j]=(arr[i][j]+x.arr[i][j])%10;

}

}

return ret;

**}**

**};**

int N , k ;

matrix I ;

matrix rec( matrix A , int n ) {

matrix ret = matrix(N) ;

matrix tmp = matrix(N) ;

matrix p = matrix(N) ;

matrix q = matrix(N) ;

matrix t = matrix(N) ;

if(n == 0 )return I ;

if( n == 1 )return A ;

ret = A^(n/2);

tmp = ret + I ;

q = rec( A , n/2 ) ;;

p = tmp \* q ;

if( n & 1 ) {

t = A^n ;

return p +t ;

}

return p ;

}

**Gaussian Ellimination**

**/\***

1. Set row and col of mat

2. Call rank() to perform gauss-elimination and find rank

3. Call isValid() to find if solution exists.

Careful about int a[x][x]. If mod^2 crosses int, take vlong

If mod is 2, it is better to use XOR since it a lot faster.

**\*/**

**struct GAUSS {**

int row, col;

vlong a[x][x];

int mod;

bool valid;

**GAUSS() {**

mod = xyz;

**}**

**void clear () {**

memset ( a, 0, sizeof a );

**}**

**void isValid ( int st ) {**

int i;

valid = true;

for ( i = st; i < row; i++ ) {

if ( a[i][col-1] ) {

valid = false;

return;

}

}

**}**

///Return Rank of Matrix

///Free variable = Variable - Rank or Col - Rank - 1

**int rank() {**

int i = 0, j = 0, k, r, u;

while(i < row && j < col - 1) {

r = i;

for(k = i; k < row; k++)

if(a[k][j]) {

r = k; ///Find non-zero coefficient

break;

}

if(a[r][j]) {

if(r != i) ///Swap row if required

for(k = 0; k < col; k++)

swap(a[r][k], a[i][k]);

///Neutralize if required. Depends on whether double or modular division

vlong v = a[i][j];

v = modInv ( v, mod );

for ( u = j; u < col; u++ ) {

a[i][u] = ( a[i][u] \* v ) % mod;

}

/\*

double v = a[i][j];

for ( u = j; u < col; u++ ) {

a[i][u] /= v;

}

\*/

for(u = i + 1; u < row; u++)

if(a[u][j]) { ///Eliminate

int v = a[u][j];

for(k = j; k < col; k++) {

a[u][k] = ( ( a[i][k] \* v ) - a[u][k] ) % mod; ///Change here if no mod

if ( a[u][k] < 0 ) a[u][k] += mod;

}

}

i++;

}

j++;

}

return i;

**}**

**void print() {**

FOR(i,0,row-1) {

FOR(j,0, col-1) {

printf ( "%d ", a[i][j] );

}

nl;

}

}

**} mat;**

**Number Theory Hn**

**void precalc() {**

arr[0] = 0 ;

arr[1] = 1 ;

for( int i = 2 ; i < 1000000 ; i++ ) {

arr[i] = arr[i-1]+ (1/double(i)) ;

}

**}**

**#define gamma 0.57721566490153286060651209008240243104215933593992**

**double Hn( double N ) {**

if( N < 1000000 ) {

return arr[int(N)];

} else return (log(double(N))+log(double(N+1)))/2 + gamma ; ;

**}**

**---------------------------------**

**long long H( int n ) {**

long long res = 0;

for( int i = 1; i <= n; i++ )

res = res + n / i;

return res;

**}**

long long a ;

scanf("%lld",&a ) ;

long long ans = 0 ;

long long LIM = sqrt( a ) ;

for( int i = 1 ; i <= LIM ; i++ ) {

ans += a/i ;

}

printf("Case %d: %lld\n",caseno++ , (ans<<1) - LIM\*LIM ) ;

**Josephous**

**/\***

The first one is for K = 2 and the second one is general.

Note: first function returns 1 based index while second one is 0 based.

**\*/**

**int f(int n) {**

if(n == 1) return 1;

return (f((n-(n&1))>>1)<<1) + ((n&1)?1:-1);

**}**

**int f(int n, int k) {**

if(n == 1) return 0;

return (f(n-1, k) + k)%n;

**}**

**Pollard Rho**

**#include<bits/stdc++.h>**

using namespace std;

#define LL long long

int p[5500], pt = 0;

**void sieve() {**

int mark[46340] = {};

int i, j;

for(i = 2; i < 46340; i++) {

if(mark[i] == 0) {

p[pt++] = i;

for(j = i+i; j < 46340; j += i)

mark[j] = 1;

}

}

**}**

**LL modmultiply(LL a,LL b,LL c) {**

LL x = 0,y = a%c;

while(b > 0) {

if(b%2 == 1) {

x = (x+y)%c;

}

y = (y\*2)%c;

b /= 2;

}

return x%c;

**}**

**LL modpow(LL x, LL y, LL mod) {**

LL ret = 1;// ret = x^y%mod;

while(y) {

if(y&1)

//ret = (ret\*x)%mod;

ret = modmultiply(ret, x, mod);

//x = (x\*x)%mod;

x = modmultiply(x, x, mod);

y >>= 1;

}

return ret;

**}**

**int isprime(LL n) {**

if(n == 2 || n == 3)

return 1;

if(n < 2 || (n&1) == 0)

return 0;

int i, a;

for(i = 0; i < 5; i++) {

a = rand()%(n-4)+2;

if(modpow(a, n-1, n) != 1)

return 0;

}

return 1;

**}**

**LL gcd(LL x, LL y) {**

if(!x) return y;

if(!y) return x;

if(x < 0) x = -x;

if(y < 0) y = -y;

LL t;

while(x%y)

t = x, x = y, y = t%y;

return y;

**}**

**vector<LL> ret;**

**LL pollard\_rho(LL n, LL c) {**

long long x = 2, y = 2;

do {

//x = (x\*x+c)%n;

x = (modmultiply(x, x, n)+c)%n;

//y = (y\*y+c)%n, y = (y\*y+c)%n;

y = (modmultiply(y, y, n)+c)%n;

y = (modmultiply(y, y, n)+c)%n;

LL d = gcd(x-y, n);

if(d > 1) return d;

} while(true);

return n;

**}**

**void small\_factorize(LL n) {**

int i;

for(i = 0; i < pt && p[i]\*p[i] <= n; i++) {

if(n%p[i] == 0) {

while(n%p[i] == 0)

ret.push\_back(p[i]), n /= p[i];

}

}

if(n != 1)

ret.push\_back(n);

**}**

**void factorize(LL n) {**

if(n == 1) return;

if(isprime(n)) {

ret.push\_back(n);

return;

}

if(n < 1000000000) {

small\_factorize(n);

return;

}

int c;

LL d = n;

for(c = 2; d == n; c++) {

d = pollard\_rho(n, c);

}

factorize(d);

factorize(n/d);

**}**

**int main() {**

sieve();

int cases ;

scanf("%d",&cases ) ;

while( cases-- ) {

long long n ;

scanf("%lld", &n);

ret.clear();

factorize(n);

sort(ret.begin(), ret.end());

cout << n << " = " ;

/\*

for( int i = 0 ; i < ret.size() ; i++ )cout << ret[i] << " " ;

cout << "\n" ;

\*/

int cnt = 1 ;

for( int i = 1 ; i < ret.size() ; i++ ){

if( ret[i] == ret[i-1] )cnt ++ ;

else {

cout << ret[i-1] ;

if( cnt > 1 ){

cout << "^" << cnt ;

}

cout << " \* " ;

cnt = 1 ;

}

}

cout << ret[ret.size()-1] ;

if( cnt > 1 )cout << "^" << cnt ;

cout << "\n" ;

}

return 0;

**}**

**Segmented Sieve**

/\*

Generates primes within interval [a, b] when b - a <= 100000

and 1 <= a <= b <= 2147483647

\*/

int base[MAX>>6], segment[RNG>>6], primes[LEN], prlen;

#define chkC(x,n) (x[n>>6]&(1<<((n>>1)&31)))

#define setC(x,n) (x[n>>6]|=(1<<((n>>1)&31)))

**void sieve() {**

int i, j, k;

for(i=3; i<LMT; i+=2) if(!chkC(base, i)) for(j=i\*i, k=i<<1; j<MAX; j+=k) setC(base, j);

for(i=3, prlen=0; i<MAX; i+=2) if(!chkC(base, i)) primes[prlen++] = i;

**}**

**int segmented\_sieve(int a, int b) {**

int rt, i, k, cnt = (a<=2 && 2<=b)? 1 : 0;

if(b<2) return 0;

if(a<3) a = 3;

if(a%2==0) a++;

memset(segment, 0, sizeof segment);

for(i=0, rt=(int)sqrt((double)b); i < prlen && primes[i] <= rt; i++) {

unsigned j = primes[i] \* ( (a+primes[i]-1) / primes[i] );

if(j%2==0) j += primes[i];

for(k=primes[i]<<1; j<=b; j+=k) if(j!=primes[i]) setC(segment, (j-a));

}

for(i=0; i<=b-a; i+=2) if(!chkC(segment, i)) cnt++;

return cnt;

**}**

**Shank’s Baby Step Giant Step**

/\*

Shanks baby step giant step - discrete logarithm algorithm

for the equation: b = a^x % p where a, b, p known, finds x

works only when p is an odd prime

\*/

**int shank(int a, int b, int p) {**

int i, j, m;

long long c, aj, ami;

map< long long, int > M;

map< long long, int > :: iterator it;

m = (int)ceil(sqrt((double)(p)));

M.insert(make\_pair(1, 0));

for(j = 1, aj = 1; j < m; j++) {

aj = (aj \* a) % p;

M.insert(make\_pair(aj, j));

}

ami = modexp(modinv(a, p), m, p);

for(c = b, i = 0; i < m; i++) {

it = M.find(c);

if(it != M.end()) return i \* m + it->second;

c = (c \* ami) % p;

}

return 0;

**}**

**Extended Euclid/ Linear Diophantine Solution**

**typedef long long Long;**

**void Egcd( Long a, Long b, Long &x, Long &y, Long &g ){**

if( !b ) x = 1,y = 0,g = a;

else{

Long x1,y1;

Egcd( b,a%b,x1,y1,g );

x = y1;

y = x1 - a/b\*y1;

}

**}**

**Long MyFloor( Long a, Long b ) {**

Long c = a / b;

if( (a%b) and a<0 ) c--;

return c;

**}**

**Long Solve( Long a, Long b, Long c, Long x1, Long x2, Long y1, Long y2 ){**

Long x,y,g;

if( a < 0 ) a \*= -1, x1 \*= -1, x2 \*= -1, swap(x1, x2);

if( b < 0 ) b \*= -1, y1 \*= -1, y2 \*= -1, swap(y1, y2);

if( !a and !b ) return !c ? ( x2-x1+1 )\*( y2-y1+1 ) : 0;

if( b==0 ){

if( c%a ) return 0;

x = c/a;

return ( x>=x1 and x<=x2)\*( y2-y1+1 );

}

if( a==0 ) {

if( c%b ) return 0;

y = c/b;

return ( y>=y1 and y<=y2 )\*( x2-x1+1 );

}

Egcd( a,b,x,y,g );

if( c%g ) return 0;

a /= g; b /= g; c /= g; g = 1;

x = x\*c; y = y\*c;

Long n2 = min( MyFloor( x-x1, b ), MyFloor( y2-y, a ) );

Long n1 = -min( MyFloor( y-y1, a ), MyFloor( x2-x, b ) );

return ( n2<n1 ) ? 0 : n2-n1+1;

**}**

**Articulation Point**

/\*

G[][]: undirected graph

cut[v] is true if node v is an articulation point / cut-vertex

\*/

vector< int > G[MAX];

int low[MAX], vis[MAX], used[MAX], cut[MAX], dfstime;

**void dfs(int u, int par = -1) {**

int i, v, child = 0;

used[u] = 1;

vis[u] = low[u] = ++dfstime;

for(i = 0; i < G[u].size(); i++) {

v = G[u][i];

if(v == par) continue;

if(used[v]) low[u] = min(low[u], vis[v]);

else {

child++;

dfs(v, u);

low[u] = min(low[u], low[v]);

if(low[v] >= vis[u]) cut[u] = 1;

}

}

if(par == -1) cut[u] = (child > 1);

**}**

**Articulation Bridge**

/\*

G[][]: undirected graph

finds all the bridges in a connected graph and

adds those edges to the Bridges[] vector

\*/

vector< int > G[MAX];

vector< pair< int, int > > Bridges;

int low[MAX], vis[MAX], used[MAX], dfstime;

**void dfs(int u, int par) {**

int i, v;

used[u] = 1;

vis[u] = low[u] = ++dfstime;

for(i = 0; i < G[u].size(); i++) {

v = G[u][i];

if(v == par) continue;

if(used[v]) low[u] = min(low[u], vis[v]);

else {

dfs(v, u);

low[u] = min(low[u], low[v]);

if(low[v] > vis[u]) Bridges.push\_back(make\_pair(u, v));

}

}

**}**

**Stable Marriage**

/\*

INPUT:

m: number of man, n: number of woman (must be at least as large as m)

L[i][]: the list of women in order of decreasing preference of man i

R[j][i]: the attractiveness of i to j.

OUTPUTS:

L2R[]: the mate of man i (always between 0 and n-1)

R2L[]: the mate of woman j (or -1 if single)

man priority

\*/

int m, n, L[MAXM][MAXW], R[MAXW][MAXM], L2R[MAXM], R2L[MAXW], p[MAXM];

**void stableMarriage() {**

int i, man, wom, hubby;

SET(R2L); CLR(p);

for(i = 0; i < m; i++ ) {

man = i;

while(man >= 0) {

while(true) {

wom = L[man][p[man]++];

if(R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]]) break;

}

hubby = R2L[wom];

R2L[L2R[man] = wom] = man;

man = hubby;

}

}

**}**

**Max Flow Dinitz**

#include<bits/stdc++.h>

using namespace std;

**struct edge{**

int v,ri;

long long c;

edge(){}

edge(int \_v,long long \_c,int \_ri){v=\_v,c=\_c,ri=\_ri;}

edge(int \_v,long long \_c){v=\_v,c=\_c;}

**};**

vector<edge>g[5003];

int source,sink,level[5003];

**void addEdge(int u,int v,long long c){**

//cout<<"addedge "<<u<<" "<<v<<" "<<c<<endl;

g[u].push\_back(edge(v,c,g[v].size()));

g[v].push\_back(edge(u,0,g[u].size()-1));

return;

**}**

**bool bfs(){**

memset(level,-1,sizeof(level));

level[source]=0;

queue<int>q;

q.push(source);

while(!q.empty()){

int src=q.front();

q.pop();

for(int i=0;i<(int)g[src].size();i++){

if(level[g[src][i].v]==-1 && g[src][i].c>0){

level[g[src][i].v]=level[src]+1;

q.push(g[src][i].v);

}

}

}

return level[sink]!=-1;

**}**

**long long dfs(int src,long long minCap){**

if(src==sink)return minCap;

long long x=0,y=0;

for(int i=0;i<(int)g[src].size();i++){

if(g[src][i].c<=0 || level[g[src][i].v]!=level[src]+1)continue;

y=dfs(g[src][i].v,min(g[src][i].c,minCap-x));

x+=y;

g[src][i].c-=y;

g[g[src][i].v][g[src][i].ri].c+=y;

if(x==minCap)break;

}

if(x==0)level[src]=0;

return x;

**}**

**int main(){**

ios\_base::sync\_with\_stdio(0);

cin.tie(nullptr);

int n,m,u,v,c;

cin>>n>>m;

source=1,sink=n;

for(int i=0;i<m;i++){

cin>>u>>v>>c;

addEdge(u,v,c);

addEdge(v,u,c);

}

long long flow=0;

while(bfs()){

flow+=dfs(source,1LL<<50);

}

cout<<flow<<"\n";

return 0;

**}**

**BPM**

**/\***

G[] is the left-side graph, must be bipartite

match(n): n is the number of nodes in left-side set

and returns the maximum possible matching.

Left[] anf Right[] ar assigned with corresponding matches

**\*/**

vector < int > G[MAX];

bool visited[MAX];

int Left[MAX], Right[MAX];

**bool dfs(int u) {**

if(visited[u]) return false;

visited[u] = true;

int len = G[u].size(), i, v;

for(i=0; i<len; i++) {

v = G[u][i];

if(Right[v]==-1) {

Right[v] = u, Left[u] = v;

return true;

}

}

for(i=0; i<len; i++) {

v = G[u][i];

if(dfs(Right[v])) {

Right[v] = u, Left[u] = v;

return true;

}

}

return false;

**}**

**int match(int n) {**

int i, ret = 0;

bool done;

SET(Left); SET(Right);

do {

done = true; CLR(visited);

for(i=0; i<n; i++) {

if(Left[i]==-1 && dfs(i)) {

done = false;

}

}

} while(!done);

for(i=0; i<n; i++) ret += (Left[i]!=-1);

return ret;

**}**

**/\***

HopCroft Karp

n: number of nodes on left side, nodes are numbered 1 to n

m: number of nodes on right side, nodes are numbered n+1 to n+m

G = NIL[0] ? G1[G[1---n]] ? G2[G[n+1---n+m]]

**\*/**

**bool bfs() {**

int i, u, v, len;

queue< int > Q;

for(i=1; i<=n; i++) {

if(match[i]==NIL) {

dist[i] = 0;

Q.push(i);

}

else dist[i] = INF;

}

dist[NIL] = INF;

while(!Q.empty()) {

u = Q.front(); Q.pop();

if(u!=NIL) {

len = G[u].size();

for(i=0; i<len; i++) {

v = G[u][i];

if(dist[match[v]]==INF) {

dist[match[v]] = dist[u] + 1;

Q.push(match[v]);

}

}

}

}

return (dist[NIL]!=INF);

**}**

**bool dfs(int u) {**

int i, v, len;

if(u!=NIL) {

len = G[u].size();

for(i=0; i<len; i++) {

v = G[u][i];

if(dist[match[v]]==dist[u]+1) {

if(dfs(match[v])) {

match[v] = u;

match[u] = v;

return true;

}

}

}

dist[u] = INF;

return false;

}

return true;

**}**

**int hopcroft\_karp() {**

int matching = 0, i;

CLR(match);

while(bfs())

for(i=1; i<=n; i++)

if(match[i]==NIL && dfs(i))

matching++;

return matching;

**}**

**Min Cost Max Flow**

#include<bits/stdc++.h>

using namespace std;

int dirr[]={0,1};

int dirc[]={1,0};

**struct edge{**

int v,c,w,ri;

edge(){}

edge(int \_v,int \_c,int \_w,int \_ri){v=\_v,c=\_c,w=\_w,ri=\_ri;}

**};**

vector<edge>g[1853];

int source,sink,pr[1853],\_index[1853],vis[1853];

**void addEdge(int u,int v,int c,int w){**

g[u].push\_back(edge(v,c,w,g[v].size()));

g[v].push\_back(edge(u,0,-w,g[u].size()-1));

return;

**}**

#define in\_node(a) (a<<1)

#define out\_node(a) (in\_node(a)|1)

**bool BF(){**

for(int i=0;i<=sink;i++){

pr[i]=-1;

vis[i]=1<<28;

}

vis[source]=0;

pr[source]=-2;

for(int k=0,done=0;done<1 && k<=sink;k++){

done=1;

for(int i=0;i<=sink;i++){

if(vis[i]==(1<<28))continue;

for(int j=0;j<(int)g[i].size();j++){

if(vis[g[i][j].v]>vis[i]+g[i][j].w && g[i][j].c>0){

vis[g[i][j].v]=vis[i]+g[i][j].w;

pr[g[i][j].v]=i;

\_index[g[i][j].v]=j;

done=0;

}

}

}

}

return (vis[sink]<(1<<28));

**}**

**int main(){**

ios\_base::sync\_with\_stdio(0);

cin.tie(nullptr);

int n;

while(cin>>n){

int a,x=0;

for(int i=0;i<1853;i++)g[i].clear();

source=2,sink=(n\*n\*2)+1;

for(int i=1;i<=n;i++){

for(int j=1;j<=n;j++){

int ii=(i-1)\*n+j;

cin>>a;

if((i==1 && j==1) || (i==n && j==n)){

addEdge(in\_node(ii),out\_node(ii),2,-a);

x+=a;

}

else addEdge(in\_node(ii),out\_node(ii),1,-a);

}

}

for(int i=1;i<=n;i++){

for(int j=1;j<=n;j++){

int ii=(i-1)\*n+j;

for(int k=0;k<2;k++){

int x=i+dirr[k],y=j+dirc[k];

if(x<1 || y<1 || x>n || y>n)continue;

int jj=(x-1)\*n+y;

addEdge(out\_node(ii),in\_node(jj),1,0);

}

}

}

int ans=0,flow=0;

while(flow<2 && BF()){

flow++;

for(int y=sink;pr[y]>0;y=pr[y]){

g[pr[y]][\_index[y]].c--;

g[y][g[pr[y]][\_index[y]].ri].c++;

}

//cout<<vis[sink]<<"\n";

ans+=vis[sink];

}

ans+=x;

cout<<-ans<<"\n";

}

return 0;

**}**

**Max Flow with Edge Demand**

**struct edge{**

int v,ri,ii;

long long ch,cl;

edge(){}

edge(int \_v,long long \_ch,long long \_cl,int \_ri,int \_ii){v=\_v,ch=\_ch,cl=\_cl,ri=\_ri,ii=\_ii;}

**};**

vector<edge>g[210];

int source,sink,/\*specialSource,\*/level[210],cnt[210],n,m,u,v,l,c/\*specialSink\*/;

**inline void addEdge(int u,int v,long long ch,long long cl,int ii){**

g[u].push\_back(edge(v,ch,cl,g[v].size(),ii));

g[v].push\_back(edge(u,0,0,g[u].size()-1,-1));

return;

**}**

**inline bool bfs(){**

memset(level,-1,sizeof(level));

level[source]=0;

queue<int>q;

q.push(source);

while(!q.empty()){

int s=q.front();

q.pop();

for(int i=0;i<(int)g[s].size();i++){

if(level[g[s][i].v]==-1 && g[s][i].ch>0){

level[g[s][i].v]=level[s]+1;

q.push(g[s][i].v);

}

}

}

return (level[sink]!=-1);

**}**

**inline long long dfs(int src,long long minCap){**

if(src==sink)return minCap;

long long x=0,y;

for(int i=0;i<(int)g[src].size();i++){

if(g[src][i].ch<=0 || level[g[src][i].v]!=level[src]+1)continue;

y=dfs(g[src][i].v,min(minCap-x,g[src][i].ch));

x+=y;

g[src][i].ch-=y;

g[g[src][i].v][g[src][i].ri].ch+=y;

if(x==minCap)break;

}

if(x==0)level[src]=0;

return x;

**}**

**int main(){**

int t;

t=RI();//scanf("%d",&t);

for(int z=1;z<=t;z++){

n=RI(),m=RI();//scanf("%d%d",&n,&m);

for(int i=0;i<210;i++){

g[i].clear();

}

memset(cnt,0,sizeof(cnt));

source=0,sink=n+1;//,specialSource=n+2,specialSink=n+3;

for(int i=0;i<m;i++){

u=RI(),v=RI(),l=RI(),c=RI();//scanf("%d%d%d%d",&u,&v,&l,&c);

addEdge(u,v,c-l,l,i);

cnt[u]-=l;

cnt[v]+=l;

}

long long sum=0;

//addEdge(source,1,INT\_MAX,0,-1);

//addEdge(n,sink,INT\_MAX,0,-1);

for(int i=1;i<=n;i++){

//cout<<"m at i = "<<i<<" is "<<cnt[i]<<"\n";

if(cnt[i]>0){

sum+=cnt[i];

addEdge(source,i,cnt[i],0,-1);

}else addEdge(i,sink,-cnt[i],0,-1);

}

//addEdge(source,specialSource,sum,0,-1);

//addEdge(specialSink,sink,sum,0,-1);

addEdge(n,1,INT\_MAX,0,-1);

long long flow=0;

while(bfs()){

flow+=dfs(source,INT\_MAX);

}

//cout<<flow<<"\n";

//cout<<g[source].size()<<" -- "<<g[sink].size()<<endl;

if(flow!=sum){

cout<<"Case "<<z<<": no\n";

continue;

}

cout<<"Case "<<z<<": yes\n";

vector<long long>ans(m);

for(int i=1;i<=n;i++){

for(int j=0;j<(int)g[i].size();j++){

if(g[i][j].ii>=0){

ans[g[i][j].ii]=g[i][j].cl+g[g[i][j].v][g[i][j].ri].ch;

}

}

}

for(int i=0;i<m;i++){

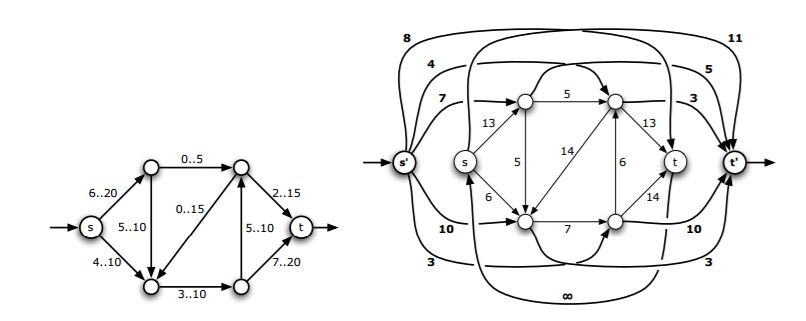
cout<<ans[i]<<"\n";

}

}

return 0;

**}**

****

**Stack Histrogram**

**/\***

**Finds largest rectangular area in a histogram in O(n)**

**\*/**

**i64 calc(int \*ht, int n) {**

i64 ret = 0;

int top = 1, st[MAX], i;

ht[0] = st[0] = ht[++n] = 0;

for(i = 1; i <= n; i++) {

while(top > 1 && ht[st[top-1]] >= ht[i]) {

ret = \_max(ret, (i64)ht[st[top-1]]\*(i64)(i - st[top-2]-1));

top--;

}

st[top++] = i;

}

return ret;

}

/\*

\* Algorithm : Hungarian algorithm

\* Max Weighted Bi-partite Matching

\* Complexity : O( N^3 )

\* Note : 0 base indexing

\*/

#include<stdio.h>

#include<string.h>

#include<vector>

#include<algorithm>

using namespace std;

#define MAX 107 // Max number of vertices in one part

#define INF 100000000 // Just infinity

long cost[MAX][MAX]; // cost matrix

long N,max\_match; // N workers and N jobs

long lx[MAX], ly[MAX]; // Labels of X and Y parts

long xy[MAX]; // xy[x] - vertex that is matched with x,

long yx[MAX]; // yx[y] - vertex that is matched with y

bool S[MAX], T[MAX]; // Sets S and T in algorithm

long slack[MAX];

long slackx[MAX]; // slackx[y] such a vertex, that

// l(slackx[y]) + l(y) - w(slackx[y],y) = slack[y]

long Prev[MAX]; // Array for memorizing alternating paths

void Init\_Labels() {

memset(lx, 0, sizeof(lx));

memset(ly, 0, sizeof(ly));

long x,y;

for( x=0; x<N; x++ ) for( y=0; y<N; y++ ) lx[x] = max(lx[x], cost[x][y]);

}

void Update\_Labels() {

long x, y, delta = INF;

for( y=0; y<N; y++ ) if(!T[y]) delta = min(delta, slack[y]);

for( x=0; x<N; x++ ) if(S[x]) lx[x] -= delta;

for( y=0; y<N; y++ ) if(T[y]) ly[y] += delta;

for( y=0; y<N; y++ ) if(!T[y]) slack[y] -= delta;

}

void Add\_To\_Tree(long x, long prevx) {

S[x] = true;

Prev[x] = prevx;

long y;

for( y=0; y<N; y++ )

if (lx[x] + ly[y] - cost[x][y] < slack[y]) {

slack[y] = lx[x] + ly[y] - cost[x][y];

slackx[y] = x;

}

}

void Augment() {

if (max\_match == N) return;

long x, y, root;

long q[MAX], wr = 0, rd = 0;

memset(S, false, sizeof(S));

memset(T, false, sizeof(T));

memset(Prev, -1, sizeof(Prev));

for( x=0; x<N; x++ ) if (xy[x] == -1) {

q[wr++] = root = x;

Prev[x] = -2;

S[x] = true;

break;

}

for( y=0; y<N; y++ ) {

slack[y] = lx[root] + ly[y] - cost[root][y];

slackx[y] = root;

}

while( true ) {

while (rd < wr) {

x = q[rd++];

for( y=0; y<N; y++ ) {

if (cost[x][y] == lx[x] + ly[y] && !T[y]) {

if (yx[y] == -1) break;

T[y] = true;

q[wr++] = yx[y];

Add\_To\_Tree(yx[y], x);

}

}

if (y < N) break;

}

if (y < N) break;

Update\_Labels();

wr = rd = 0;

for( y=0; y<N; y++ ) {

if (!T[y] && slack[y] == 0) {

if (yx[y] == -1) {

x = slackx[y];

break;

} else {

T[y] = true;

if (!S[yx[y]]) {

q[wr++] = yx[y];

Add\_To\_Tree(yx[y], slackx[y]);

}

}

}

}

if (y < N) break;

}

if (y < N) {

max\_match++;

for (long cx = x, cy = y, ty; cx != -2; cx = Prev[cx], cy = ty) {

ty = xy[cx];

yx[cy] = cx;

xy[cx] = cy;

}

Augment();

}

}

long Hungarian() {

long x,ret = 0;

max\_match = 0;

memset(xy, -1, sizeof(xy));

memset(yx, -1, sizeof(yx));

Init\_Labels();

Augment();

for( x=0; x<N; x++ ) ret += cost[x][xy[x]];

return ret;

}

/\*

\* Algorithm : Stoer Wagner all pair Min Cut

\* Complextity : O( n^3 )

\* Note : All vertex is 1 based indexing

\*/

#include<stdio.h>

#include<algorithm>

using namespace std;

#define MAX\_V 1007

#define INF 777777777

long nV,nE;

long Cap[MAX\_V+7][MAX\_V+7];

long Dist[MAX\_V+7],Done[MAX\_V+7];

long Best,Ind;

long AllPairMinCut( void ) {

long i,j,u,v,Ans = INF,N = nV;

while( N > 1 ) {

memset( Dist,0,sizeof(Dist));

memset( Done,0,sizeof(Done));

for( i=1; i<=N; i++) {

Best = Ind = -1;

for( j=1; j<=N; j++ ) {

if(!Done[j] && Dist[j] > Best) {

Best = Dist[j];

Ind = j;

}

}

if( i+1==N ) u = Ind;

if( i==N ) {

v = Ind;

Ans = min( Ans,Best );

}

Done[Ind] = 1;

for( j=1; j<=N; j++) {

Dist[j] += Cap[Ind][j];

}

}

if( u > v) swap( u,v);

for( i=1; i<=N; i++ ) {

Cap[u][i] += Cap[v][i];

Cap[i][u] += Cap[i][v];

}

for( i=1; i<=N; i++) {

Cap[v][i] = Cap[N][i];

Cap[i][v] = Cap[i][N];

}

N--;

}

return Ans;

}

int main( void) {

memset(Cap,0,sizeof(Cap));

return 0;

}

**Sparse Table**

#define MAX 100000+5

#define LGMAX 18

int ST[MAX][LGMAX];

void rmq( int n ){

for( int j = 1 ; 1<<j <= n ; j++ ){

for( int i = 0 ; i+(1<<j)-1 < n ; i++ ){

ST[i][j] = min( ST[i][j - 1] , ST[i+(1<<(j-1))][j - 1] ) ;

}

}

}

**int main() {**

int cases = readInt() ;

int caseno = 1 ;

while( cases -- ){

int N = readInt() ;

int Q = readInt() ;

for( int i = 0 ; i < N ; i++ ){

ST[i][0] = readInt() ;

}

rmq(N) ;

cout << "Scenario #" << caseno++ << ":\n" ;

while( Q -- ){

int x = readInt()-1 ;

int y = readInt()-1 ;

int lim = y-x+1 , lg = 0 ;

while( lim ){

lim >>= 1 ;

lg++ ;

}

lg-- ;

cout << min( ST[x][lg] , ST[y-(1<<lg)+1][lg] ) << "\n" ;

}

}

return 0 ;

**}**

**BIG INT**

const int base = 1000000000;

const int base\_digits = 9;

**struct bigint {**

vector<int> a;

int sign;

**bigint() :**

sign(1) {

**}**

**bigint(long long v) {**

\*this = v;

**}**

**bigint(const string &s) {**

read(s);

**}**

**void operator=(const bigint &v) {**

sign = v.sign;

a = v.a;

**}**

**void operator=(long long v) {**

sign = 1;

if (v < 0)

sign = -1, v = -v;

for (; v > 0; v = v / base)

a.push\_back(v % base);

**}**

**bigint operator+(const bigint &v) const {**

if (sign == v.sign) {

bigint res = v;

for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.size()) || carry; ++i) {

if (i == (int) res.a.size())

res.a.push\_back(0);

res.a[i] += carry + (i < (int) a.size() ? a[i] : 0);

carry = res.a[i] >= base;

if (carry)

res.a[i] -= base;

}

return res;

}

return \*this - (-v);

**}**

**bigint operator-(const bigint &v) const {**

if (sign == v.sign) {

if (abs() >= v.abs()) {

bigint res = \*this;

for (int i = 0, carry = 0; i < (int) v.a.size() || carry; ++i) {

res.a[i] -= carry + (i < (int) v.a.size() ? v.a[i] : 0);

carry = res.a[i] < 0;

if (carry)

res.a[i] += base;

}

res.trim();

return res;

}

return -(v - \*this);

}

return \*this + (-v);

**}**

**void operator\*=(int v) {**

if (v < 0)

sign = -sign, v = -v;

for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {

if (i == (int) a.size())

a.push\_back(0);

long long cur = a[i] \* (long long) v + carry;

carry = (int) (cur / base);

a[i] = (int) (cur % base);

//asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"(base));

}

trim();

**}**

**bigint operator\*(int v) const {**

bigint res = \*this;

res \*= v;

return res;

**}**

**friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1) {**

int norm = base / (b1.a.back() + 1);

bigint a = a1.abs() \* norm;

bigint b = b1.abs() \* norm;

bigint q, r;

q.a.resize(a.a.size());

for (int i = a.a.size() - 1; i >= 0; i--) {

r \*= base;

r += a.a[i];

int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];

int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];

int d = ((long long) base \* s1 + s2) / b.a.back();

r -= b \* d;

while (r < 0)

r += b, --d;

q.a[i] = d;

}

q.sign = a1.sign \* b1.sign;

r.sign = a1.sign;

q.trim();

r.trim();

return make\_pair(q, r / norm);

**}**

**bigint operator/(const bigint &v) const {**

return divmod(\*this, v).first;

**}**

**bigint operator%(const bigint &v) const {**

return divmod(\*this, v).second;

**}**

**void operator/=(int v) {**

if (v < 0)

sign = -sign, v = -v;

for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {

long long cur = a[i] + rem \* (long long) base;

a[i] = (int) (cur / v);

rem = (int) (cur % v);

}

trim();

**}**

**bigint operator/(int v) const {**

bigint res = \*this;

res /= v;

return res;

**}**

**int operator%(int v) const {**

if (v < 0)

v = -v;

int m = 0;

for (int i = a.size() - 1; i >= 0; --i)

m = (a[i] + m \* (long long) base) % v;

return m \* sign;

**}**

**void operator+=(const bigint &v) {**

\*this = \*this + v;

**}**

**void operator-=(const bigint &v) {**

\*this = \*this - v;

**}**

**void operator\*=(const bigint &v) {**

\*this = \*this \* v;

**}**

**void operator/=(const bigint &v) {**

\*this = \*this / v;

**}**

**bool operator<(const bigint &v) const {**

if (sign != v.sign)

return sign < v.sign;

if (a.size() != v.a.size())

return a.size() \* sign < v.a.size() \* v.sign;

for (int i = a.size() - 1; i >= 0; i--)

if (a[i] != v.a[i])

return a[i] \* sign < v.a[i] \* sign;

return false;

**}**

**bool operator>(const bigint &v) const {**

return v < \*this;

**}**

**bool operator<=(const bigint &v) const {**

return !(v < \*this);

**}**

**bool operator>=(const bigint &v) const {**

return !(\*this < v);

**}**

**bool operator==(const bigint &v) const {**

return !(\*this < v) && !(v < \*this);

**}**

**bool operator!=(const bigint &v) const {**

return \*this < v || v < \*this;

**}**

**void trim() {**

while (!a.empty() && !a.back())

a.pop\_back();

if (a.empty())

sign = 1;

**}**

**bool isZero() const {**

return a.empty() || (a.size() == 1 && !a[0]);

**}**

**bigint operator-() const {**

bigint res = \*this;

res.sign = -sign;

return res;

**}**

**bigint abs() const {**

bigint res = \*this;

res.sign \*= res.sign;

return res;

**}**

**long long longValue() const {**

long long res = 0;

for (int i = a.size() - 1; i >= 0; i--)

res = res \* base + a[i];

return res \* sign;

**}**

**friend bigint gcd(const bigint &a, const bigint &b) {**

return b.isZero() ? a : gcd(b, a % b);

**}**

**friend bigint lcm(const bigint &a, const bigint &b) {**

return a / gcd(a, b) \* b;

**}**

**void read(const string &s) {**

sign = 1;

a.clear();

int pos = 0;

while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+')) {

if (s[pos] == '-')

sign = -sign;

++pos;

}

for (int i = s.size() - 1; i >= pos; i -= base\_digits) {

int x = 0;

for (int j = max(pos, i - base\_digits + 1); j <= i; j++)

x = x \* 10 + s[j] - '0';

a.push\_back(x);

}

trim();

**}**

**friend istream& operator>>(istream &stream, bigint &v) {**

string s;

stream >> s;

v.read(s);

return stream;

**}**

**friend ostream& operator<<(ostream &stream, const bigint &v) {**

if (v.sign == -1)

stream << '-';

stream << (v.a.empty() ? 0 : v.a.back());

for (int i = (int) v.a.size() - 2; i >= 0; --i)

stream << setw(base\_digits) << setfill('0') << v.a[i];

return stream;

**}**

**static vector<int> convert\_base(const vector<int> &a, int old\_digits, int new\_digits) {**

vector<long long> p(max(old\_digits, new\_digits) + 1);

p[0] = 1;

for (int i = 1; i < (int) p.size(); i++)

p[i] = p[i - 1] \* 10;

vector<int> res;

long long cur = 0;

int cur\_digits = 0;

for (int i = 0; i < (int) a.size(); i++) {

cur += a[i] \* p[cur\_digits];

cur\_digits += old\_digits;

while (cur\_digits >= new\_digits) {

res.push\_back(int(cur % p[new\_digits]));

cur /= p[new\_digits];

cur\_digits -= new\_digits;

}

}

res.push\_back((int) cur);

while (!res.empty() && !res.back())

res.pop\_back();

return res;

**}**

**typedef vector<long long> vll;**

**static vll karatsubaMultiply(const vll &a, const vll &b) {**

int n = a.size();

vll res(n + n);

if (n <= 32) {

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

res[i + j] += a[i] \* b[j];

return res;

}

int k = n >> 1;

vll a1(a.begin(), a.begin() + k);

vll a2(a.begin() + k, a.end());

vll b1(b.begin(), b.begin() + k);

vll b2(b.begin() + k, b.end());

vll a1b1 = karatsubaMultiply(a1, b1);

vll a2b2 = karatsubaMultiply(a2, b2);

for (int i = 0; i < k; i++)

a2[i] += a1[i];

for (int i = 0; i < k; i++)

b2[i] += b1[i];

vll r = karatsubaMultiply(a2, b2);

for (int i = 0; i < (int) a1b1.size(); i++)

r[i] -= a1b1[i];

for (int i = 0; i < (int) a2b2.size(); i++)

r[i] -= a2b2[i];

for (int i = 0; i < (int) r.size(); i++)

res[i + k] += r[i];

for (int i = 0; i < (int) a1b1.size(); i++)

res[i] += a1b1[i];

for (int i = 0; i < (int) a2b2.size(); i++)

res[i + n] += a2b2[i];

return res;

**}**

**bigint operator\*(const bigint &v) const {**

vector<int> a6 = convert\_base(this->a, base\_digits, 6);

vector<int> b6 = convert\_base(v.a, base\_digits, 6);

vll a(a6.begin(), a6.end());

vll b(b6.begin(), b6.end());

while (a.size() < b.size())

a.push\_back(0);

while (b.size() < a.size())

b.push\_back(0);

while (a.size() & (a.size() - 1))

a.push\_back(0), b.push\_back(0);

vll c = karatsubaMultiply(a, b);

bigint res;

res.sign = sign \* v.sign;

for (int i = 0, carry = 0; i < (int) c.size(); i++) {

long long cur = c[i] + carry;

res.a.push\_back((int) (cur % 1000000));

carry = (int) (cur / 1000000);

}

res.a = convert\_base(res.a, 6, base\_digits);

res.trim();

return res;

}

**};**

**Euler Circuit**

#include<iostream>

//every edge once

using namespace std;

int f[100]={0}, ans[100]={0};

bool g[100][100]={0}, v[100]={0};

int n=0, m=0, c=0;

**void dfs(int k)**

**{**

for (int i=1;i<=n;i++)

if (g[k][i])

{

g[k][i]=false;

g[i][k]=false;

dfs(i);

}

m++;

ans[m]=k;

**}**

**int main(void)**

**{**

cin >> n >> m;

for (int i=1;i<=m;i++)

{

int x=0, y=0;

g[x][y]=true;

g[y][x]=true;

f[x]++;

f[y]++;

}

m=0;

int k1=0;

for (int i=1;i<=n;i++)

{

if (f[i]%2==1) k1++;

if (k1>2)

{

cout << "error" << endl;

return 0;

}

if (f[i]%2 && c==0) c=i;

}

if (c==0) c=1;

dfs(x);

for (int i=m;i>=1;i--) cout << ans[i] << endl;

return 0;

**}**

**Lexicographically Smallest Cyclic Shift**

**/\***

**Finds alphabetically first representation of a cyclic string in O(length)**

**\*/**

**inline int minimumExpression(char \*s) {**

int i, j, k, n, len, p, q;

len = n = strlen(s), n <<= 1, i = 0, j = 1, k = 0;

while(i + k < n && j + k < n) {

p = i+k >= len ? s[i+k-len] : s[i+k];

q = j+k >= len ? s[j+k-len] : s[j+k];

if(p == q) k++;

else if(p > q) { i = i+k+1; if(i <= j) i = j+1; k = 0; }

else if(p < q) { j = j+k+1; if(j <= i) j = i+1; k = 0; }

}

return i < j ? i : j;

**}**

**nCr**

**void precalc() {**

for(int i = 1 ; i <= 1000 ; i++) {

nCr[i][0]=1;

nCr[i][1]=i;

for(int j = 2 ; j <= i ; j++ ) {

nCr[i][j] = ( nCr[i-1][j-1] + nCr[i-1][j] ) % MOD;

}

}

**}**

**Pow Function**

**long long Pow(long long n , long long k) {**

long long ret = 1;

while( k ) {

if( k&1 ) {

ret \*= n;

ret %= MOD;

}

n \*= n;

n %= MOD;

k >>= 1;

}

return ret;

**}**

**Zobayer bhai’s Sieve**

#define MAX 10000000

unsigned flag[MAX/64];

vector<long long>prime ;

#define chkC(n) (flag[n>>6]&(1<<((n>>1)&31)))

#define setC(n) (flag[n>>6]|=(1<<((n>>1)&31)))

int lim;

**void sieve() {**

unsigned i, j, k;

flag[0]|=0;

int sqrtN = sqrt(MAX) ;

for(i=3; i<= sqrtN ; i+=2)

if(!chkC(i))

for(j=i\*i,k=i<<1; j<MAX; j+=k)

setC(j);

prime.push\_back(2);

for(i=3; i<MAX; i+=2)

if(!chkC(i))

prime.push\_back(i) ;

lim = prime.size() ;

**}**

**Treap**

**#include <bits/stdc++.h>**

**using namespace std;**

**struct Node{**

int v,prior;

int size;

Node \*l,\*r;

Node(){}

Node(int \_v,int \_p){

v = \_v;

prior = \_p;

size = 1;

l = r = NULL;

}

**};**

**int Size(Node\* x){**

if (x == NULL) return 0;

return x->size;

**}**

**void Update(Node \*x){**

if (x == NULL) return ;

x->size = 1 + Size( x->l ) + Size( x->r );

**}**

**void Split(Node \*idx,Node\* &Left,Node\* &Right,int v){**

if (!idx){

Left = Right = NULL;

return ;

}

if (idx->v < v) {

Left = idx;

Split(idx->r,Left->r,Right,v);

}

else{

Right= idx;

Split(idx->l,Left,Right->l,v);

}

Update(Left);

Update(Right);

**}**

**Node\* Join( Node\* Left, Node\* Right ){**

if (!Left) return Right;

if (!Right) return Left;

if (Left->prior > Right->prior){

Left->r = Join(Left->l,Right);

Update(Left);

return Left;

}

else{

Right->l = Join(Left,Right->l);

Update(Right);

return Right;

}

**}**

**Node\* Insert(Node\* root,int v){**

Node \*Left,\*Right;

Split(root,Left,Right,v);

Node\* newAdd = new Node(v,rand());

root = Join( Left,newAdd );

root = Join( root,Right );

return root;

**}**

**Node\* Delete(Node\* root,int v){**

Node \*Left,\*Mid,\*Right;

Split(root,Left,Right,v);

Split(Right,Mid,Right,v+1);

root = Join(Left,Right);

return root;

**}**

**void Print(Node \*root){**

if (!root) return ;

Print(root->l);

cout << root->v << " ";

Print(root->r);

**}**

**int main(){**

Node\* root = NULL;

root = Insert(root,10);

Print(root);cout << endl;

root = Insert(root,20);

Print(root);cout << endl;

root = Insert(root,5);

Print(root);cout << endl;

root = Delete(root,10);

Print(root);cout << endl;

root = Delete(root,25);

Print(root);cout << endl;

return 0;

**}**

**#include<bits/stdc++.h>**

**using namespace std ;**

**#define MAX 1000+5**

**vector<pair<int,int> > g[MAX] ;**

**int hackenbush(int u , int par) {**

if( g[u].size() == 0 )return 0 ;

int ret = 0 ;

for( int i = 0 ; i < (int)g[u].size() ; i++ ) {

int v = g[u][i].first ;

int w = g[u][i].second ;

if( v == par )continue ;

if( w == 1 )ret ^= (hackenbush(v,u)+1);

else if( !(w&1) ) ret ^= hackenbush(v,u) ; // even-> merge

else ret ^= (hackenbush(v,u)^1) ; // odd->so assume edges reduced to 1

}

return ret ;

**}**

**int main() {**

ios\_base::sync\_with\_stdio(0) ;

cin.tie(0) ;

int cases , caseno = 1 ;

cin >> cases ;

while( cases -- ) {

int n ;

cin >> n ;

for( int i = 0 ; i < n+1 ; i++ ) {

g[i].clear() ;

}

for( int i = 0 ; i < n-1 ; i++ ) {

int u , v , w ;

cin >> u >> v >> w ;

g[u].push\_back( make\_pair(v,w) );

g[v].push\_back( make\_pair(u,w) );

}

int res = hackenbush(0,-1) ;

if( res )cout << "Case " << caseno++ << ": Emily\n" ;

else cout << "Case " << caseno++ << ": Jolly\n" ;

}

return 0 ;

**}**

**/\***

The Fusion Principle: The vertices on any circuit may be fused without changingthe Sprague-Grundy value of the graph

The colon principle states that, if a node is connected with stalks of length L0 , L1 , L2 , ... , Ln-1 then all of these stalks can be replaced with a single stalk of length L0^L1^L2^.....^Ln-1

a circuit with an odd number of edges reduces to a single edge, and a circuit with an even number of edges reduces to a single vertex

**\*/**