import java.util.Scanner;

class Q8

{

static double d,vl,hl;

//d is the separation

//vl is the vertical separation limit

//hl is the horizontal separation limit

static double p[][];

//array of points col1:x co-ordinates col2:y co-ordinates

static boolean changed[][];

//boolean array storing true if the change is made in x or y of a point

static int stack[],top=-1,count=0;

//stack is a stack that stores point positions that have a co-ordinate unchanged

//top is top of stack

//count is count of elements that have been changed

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);//for input

System.out.println("Enter d:");

d=sc.nextDouble();

// computing vl and hl

vl=1.5\*d;

hl=2\*d;

int i;//i is loop counters

//memory allocation

p=new double[(int)(150/d)+1][2];

changed=new boolean[(int)(150/d)+1][2];

stack=new int[(int)(150/d)+1];

System.out.println("Enter the coordinates of points");

//input loop

for(i=1;i<150/d;i++)

{

p[i][0]=sc.nextDouble();

p[i][1]=sc.nextDouble();

//marking all points as unchanged initially

changed[i][0]=false;

changed[i][1]=false;

}

//initializing the end cities as (0,0) as the line passes through them

p[0][0]=p[0][1]=p[(int)(150/d)][0]=p[(int)(150/d)][1]=0;

//testing x and y co-ordinates separately

//functions return number of unchanged points

count=(p.length-testx())+(p.length-testy());

System.out.println(count);

}

static int testx()

{

int region=1,i,t1,t2;

//region used as a count of regions

//i is loop counter

push(0);//pushing the first city as it can never change

for(i=1;i<p.length-1;i++)

//looping till last point on the highway other than the last city

{

//Looping through the array. If the previous unchanged point and the

//current point are separated by a distance greater than number of ranges

//times limit of separation then any one point must be changed.

//As the previously unchanged point satisfies the conditions with previous

//points, the current point is changed.

if(Math.abs(p[stack[top]][0]-p[i][0])>region\*hl)

{

changed[i][0]=true;

region++;//increasing the region which signifies number of regions

}

else

{

changed[i][0]=false;//if unchanged

push(i);//stored as last unchanged point

region=1;

}

}

//the same process is reoeated in the reverse order as some cases need further

//back tracking like 0 60 120 180 60 0. Hence the same algorithm is

//done in reverse order and minimum number of changes are considered

t1=top+1;//number of unchanged points

top=0;

for(i=p.length-1;i>0;i--)

{

if(Math.abs(p[stack[top]][0]-p[i][0])>region\*hl)

{

changed[i][0]=true;

region++;//increasing the region which signifies number of regions

}

else

{

changed[i][0]=false;//if unchanged

push(i);//stored as last unchanged point

region=1;

}

}

t2=top+1;

return t1>t2?t1:t2;

//the case that gives maximum number of unchanged points is returned

}

static int testy()

{

//exactly same as x

int region=1,i,t1,t2;

top=0;

for(i=1;i<p.length-1;i++)

{

if(Math.abs(p[stack[top]][1]-p[i][1])>region\*hl)

{

changed[i][1]=true;

region++;

}

else

{

changed[i][1]=false;

push(i);

region=1;

}

}

t1=top+1;

top=0;

for(i=p.length-1;i>0;i--)

{

if(Math.abs(p[stack[top]][1]-p[i][1])>region\*hl)

{

changed[i][1]=true;

region++;

}

else

{

changed[i][1]=false;

push(i);

region=1;

}

}

t2=top+1;

return t1>t2?t1:t2;

}

static void push(int x)

{

//basic push as stack has size of maximum number of elements that can be pushed

top++;

stack[top]=x;

}

static int pop()

{

//basic pop as stack has size of maximum number of elements that can be pushed

top--;

return stack[top+1];

}

}