University of Scranton
ACM Student Chapter / Computing Sciences Department
19th Annual High School Programming Contest (2009)

Problem 1: Fibonacci String Sequences

One of the most famous families of sequences in mathematics is the family of **Fibonacci** sequences. A Fibonacci sequence begins with two chosen values and is such that every value thereafter is the sum of the previous two. For example, if we choose to begin the sequence with 0 and 1, respectively, we get

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

The same idea can be applied to character strings. However, rather than adding two consecutive elements of a sequence to compute the next one, we concatenate them. For example, if we start with the strings `a` and `ba`, we get the sequence

`a`, `ba`, `aba`, `baaba`, `ababaaba`, `baabaababaaba`, `ababaababaabaababaaba`, ...

Develop a program that, given a positive integer m and two character strings, s_1 and s_2, displays the first m elements of the Fibonacci string sequence (as defined above) whose first two elements are s_1 and s_2, respectively.

Input: The first line contains a positive integer n indicating how many instances of the problem are described thereafter. Each instance of the problem is described on three lines, the first of which contains a positive integer m, the second of which contains a string s_1, and the third of which contains a string s_2. (Neither s_1 nor s_2 contains any spaces (or, more generally, "white space").)

Output: For each triple (m, s_1, s_2) given as input, display the first m elements of the Fibonacci string sequence that begins with s_1 and s_2, respectively, one string per line, and followed by a blank line.

Sample input and output are on next page.

```
Sample input
------------
2
6
a
ba
7
spock
kirk


Corresponding output
--------------------
a
ba
aba
baaba
ababaaba
baabaababaaba

spock
kirk
spockkirk
kirkspockkirk
spockkirkkirkspockkirk
kirkspockkirkspockkirkkirkspockkirk
spockkirkkirkspockkirkkirkspockkirkspockkirkkirkspockkirk
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
19th Annual High School Programming Contest (2009)

Problem 2: Intersecting Circles

A *circle* is the set of all points on the plane that are at a particular distance (called the *radius*) from a particular point (called the *center*).

The *intersection* of two circles is the set of points that are common to both circles. This set necessarily contains either zero, one, two, or (only in the case that the two circles are the same) infinitely many points.

Develop a program that, given two circles, determines whether or not their intersection is non-empty.

Below are two figures. The first shows two pairs of intersecting circles. The second shows two pairs of non-intersecting circles.

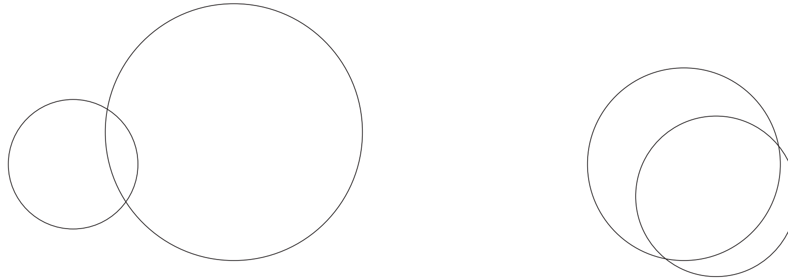

Figure 1: Two Pairs of Intersecting Circles

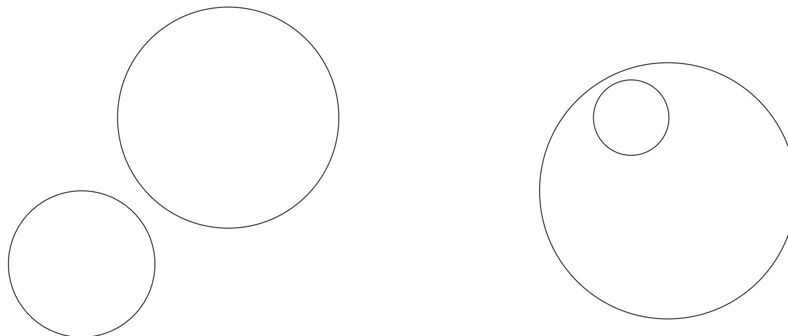

Figure 2: Two Pairs of Non-Intersecting Circles

Hint: Recall that the distance between the points (x_1, y_1) and (x_2, y_2) is

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Input: The first line contains a positive integer n indicating how many pairs of circles are to be analyzed. On each of the following n lines, a pair of circles is described. Each circle is described by three real numbers, the first of which is its radius and the last two of which are the x- and y-coordinates of its center, respectively. (Note that the sample input shown below describes the four pairs of circles in the figures.)

Output: For each pair of circles given as input, the program is to report whether or not they intersect.

```
Sample input
------------
4
2.0 3.0 4.0 4.0 8.0 5.0
3.0 0.0 0.0 2.5 1.0 -1.0
3.0 2.0 7.0 2.0 -2.0 3.0
3.5 2.5 0.0 1.0 1.5 2.0
```

```
Corresponding output
--------------------
The circles intersect.
The circles intersect.
The circles do not intersect.
The circles do not intersect.
```

Problem 3: DFA String Acceptance

Depicted in the figure below is a *deterministic finite automaton* (DFA) that we will call M. Each circle represents a *state* and each arrow labeled by a symbol represents a *transition* from one state to another (or itself) associated with that symbol. The unlabeled arrow points to the *initial state*. Double circles correspond to *accepting states*. By convention, we name the states in a DFA s_0, s_1, ..., s_{m-1}, where m is the number of states and s_0 is the initial state. The set of symbols appearing as labels on transitions is called the *alphabet* of the DFA. Notice that M's alphabet is $\{a, b\}$.

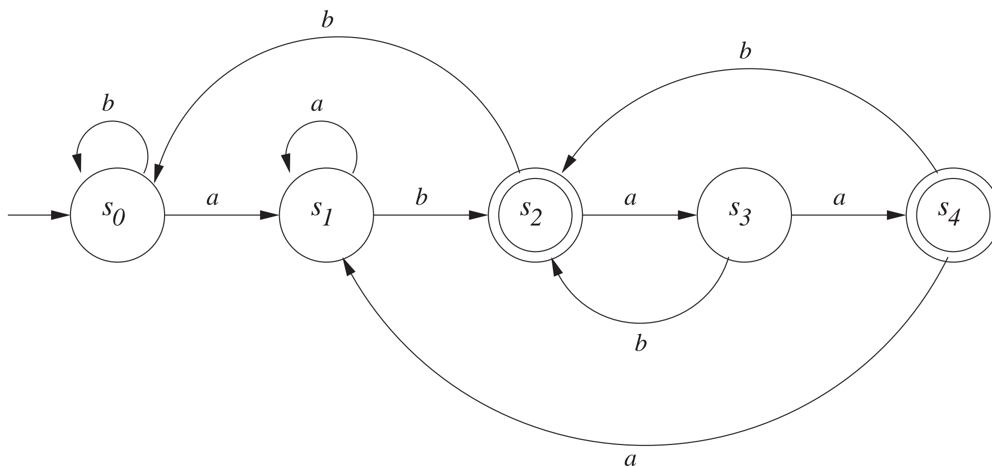

Figure 3: Deterministic Finite Automaton M

Presented with an input string, a DFA processes it as follows: Beginning in the initial state, it hops from state to state, following the sequence of transitions whose labels spell out the input string. For example, presented with the input string *aabba*, M follows the path

$$s_0 \xrightarrow{a} s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{b} s_0 \xrightarrow{a} s_1$$

and thereby finishes in state s_1. In a DFA, each state has, for each symbol of the alphabet, exactly one outgoing transition labeled by that symbol. (This is what makes it "deterministic", as opposed to "nondeterministic".) Therefore, for every state s_i and every string x, there is exactly one path beginning at s_i whose labels spell out x.

If processing a string causes a DFA to finish in one of its accepting states, we say that the automaton *accepts* that string. Otherwise, we say that the automaton *rejects* that string. In the example above, M finished in the non-accepting state s_1 after processing *aabba*; hence it rejects that string. On the other hand, M finishes in the accepting state s_4 after processing *ababaa*, and so it accepts that string.

Note: If you examine M closely, you should be able to figure out that it accepts precisely those strings composed of a's and b's that end in either ab or $abaa$. *End of note.*

Develop a program that, given a DFA with alphabet $\{a, b\}$ and some input strings composed of a's and b's, reports, for each input string, whether or not it is accepted by the DFA.

Input: The first line contains a positive integer m indicating the number of states in the DFA to be tested, followed by the number k of accepting states ($0 \leq k \leq m$) in that DFA. The second line contains a list of k distinct integers, each in the range $0..m-1$, identifying the accepting states. The next m lines describe the outgoing transitions from states s_0, s_1, ..., s_{m-1}, in that order, one state per line. On each such line will be two integers (in the range $0..m-1$) identifying the states to which the transitions labeled a and b go, respectively, from the state in question. (The sample input below describes M.)

On the following line is a positive integer r indicating how many input strings are to be processed. Each of the following r lines contains one such string.

Output: For each string given as input, generate one line of output that classifies that string as being either accepted or rejected by the DFA.

```
Sample Input
------------
5 2
2 4
1 0
1 2
3 0
4 2
1 2
3
aabba
ababaa
bbabbbbab
```

```
Corresponding Output
--------------------
aabba is rejected
ababaa is accepted
bbabbbbab is accepted
```

Problem 4: Payroll Processing

Workers who are paid an hourly wage typically submit *time cards* to document during which hours on which days they were at work. At the end of each week, the submitted time cards are used for calculating, for each employee, the number of hours he worked during that week. Using this figure, along with the employee's hourly wage, we calculate his gross pay for that week.

Develop a program that, given as input a sequence of employee records followed by a sequence of time card records, calculates each employee's gross pay. For the first 40 hours worked, an employee should be paid at a rate corresponding to his hourly wage. For any time beyond 40 hours, an employee should be paid at a rate one and one-half (i.e., 1.5) times his hourly wage.

Input: The first line contains a positive integer n, where $n \leq 20$, indicating the number of employees. Each of the following n lines contains an employee record, which includes two fields (separated by a space): the employee's name (a character string not including any white space) and his hourly wage (a real number). The records are in (ascending) alphabetical order with respect to the names.

The line following the last employee record contains a positive integer m indicating how many time cards were submitted during the week. Each of the next m lines contains a time card record, which includes two fields (separated by a space): an employee name and a positive integer indicating a number of (contiguous) minutes worked. The time card records are also ordered by name.

For a given employee, there may be zero, one, or more time cards pertaining to her. The total number of minutes worked by an employee is taken to be the sum of the figures on the time cards pertaining to that employee.

Output: For each employee, excluding those whose time worked was zero, generate a line of output that includes her name, number of hours worked, and gross pay (preceded by a dollar sign ('$')). (While rounding the numbers to two places after the decimal point (as in the sample output) is preferred, it is not mandatory.) Don't forget to account for overtime pay, as described earlier.

Sample input and output appear on the next page.

```
Sample input                  Corresponding output
------------                  --------------------
4                             Alice: 9.25 hours, $136.44
Alice 14.75                   Jim: 1.67 hours, $13.33
Bert 16.25                    Ruth: 41.00 hours, $830.00
Jim 8.00
Ruth 20.00
8
Alice 480
Alice 75
Jim 100
Ruth 500
Ruth 460
Ruth 500
Ruth 500
Ruth 500
```

Problem 5: Interval Union

Let a and b be numbers satisfying $a \leq b$. Then the *closed interval* from a to b, denoted by $[a, b]$, is the set of real numbers x satisfying $a \leq x \leq b$. Here, a is said to be the interval's *lower bound* and b is said to be its *upper bound*.

Recall that, if S and T are sets, their union, denoted $S \cup T$, is the set containing any value that is a member of either (or both of) S or T. For example,

$$\{\text{cat}, \text{dog}, \text{elk}\} \cup \{\text{bear}, \text{dog}, \text{lion}, \text{cat}, \text{gorn}\} = \{\text{cat}, \text{dog}, \text{elk}, \text{bear}, \text{lion}, \text{gorn}\}$$

Because a closed interval is a set, we can take the union of two (or more) of them, so that

$$[a_1, b_1] \cup [a_2, b_2] \cup \cdots \cup [a_m, b_m]$$

is the set of real numbers x such that $a_i \leq x \leq b_i$ for some i satisfying $1 \leq i \leq m$.

The expression above is said to be in *normal form* if $b_i < a_{i+1}$ for each i satisfying $1 \leq i < m$. That is, a union of closed intervals is in normal form if the intervals are listed so that each one's upper bound is less than the following one's lower bound. (In particular, this means that no two intervals overlap.)

For example, consider the expression

$$[3, 9] \cup [18, 25] \cup [10, 14] \cup [21, 22] \cup [7, 12]$$

To put this into normal form, we observe that $[3, 9]$ overlaps with $[7, 12]$, which overlaps with $[10, 14]$, so that the union of these three is $[3, 14]$. Also, $[18, 25]$ overlaps $[21, 22]$ (indeed, the former wholly contains the latter), and their union is $[18, 25]$. Putting the two resultant intervals in proper order, we get the normal form

$$[3, 14] \cup [18, 25]$$

Develop a program that, given as input a collection of closed intervals having integer lower and upper bounds, outputs the union of those intervals in normal form.

Input: The first line contains a positive integer n indicating the number of unions of intervals that are to be processed.

Each such union is described by a positive integer r on one line, followed by the descriptions of r closed intervals, one per line. Each closed interval is described by two integers a and b satisfying $a \leq b$.

Hint: You may find it useful, in processing a union of intervals, to sort them first. *End of hint.*

Output: For each collection of intervals given as input, the program should generate a single line of output describing the union of that collection of intervals, in normal form. In place of the $\cup$ symbol, use the plus sign (i.e., $+$).

```
Sample input
------------
4
3
3 9
18 20
7 12
7
10 15
-6 2
27 40
50 53
-4 0
4 7
12 44
5
-6 2
27 40
50 53
10 15
7 30
5
-6 2
27 40
50 53
10 15
7 45
```

```
Corresponding output
--------------------
[3,12] + [18,20]
[-6,2] + [4,7] + [10,44] + [50,53]
[-6,2] + [7,40] + [50,53]
[-6,2] + [7,45] + [50,53]
```

Problem 6: Point in Interior of Polygon?

A *polygon* is defined to be a figure on the cartesian plane formed by line segments such that

(1) each line segment intersects exactly two others, one at each endpoint, and
(2) no two line segments with a common endpoint are collinear (i.e., lie on the same line).

The endpoints of the line segments are referred to as the polygon's *vertices* (singular: *vertex*) and the line segments are referred to as its *edges* (or *sides*). A standard way of representing a polygon is as a sequence $\langle p_1, p_2, \cdots, p_m \rangle$ of vertices, where, for each i satisfying $1 \leq i < m$, line segment $\overline{p_i p_{i+1}}$ forms an edge of the polygon, as does $\overline{p_m p_1}$.

Informally, any point on the plane that lies on the bounded region enclosed by the edges of a polygon is said to be in the polygon's *interior*. (The points comprising the polygon's edges are *not* considered to be in its interior, but this will be of no concern to us.) The figure below shows an 8-sided polygon with its interior shaded and its vertices labeled by cartesian coordinates.

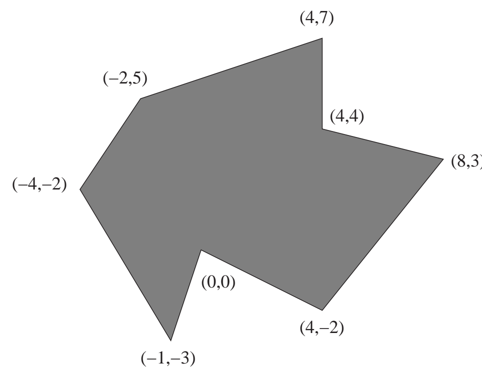

Figure 4: A Polygon with Shaded Interior

Develop a program that, given a polygon G and a point P that does not lie on any edge of G, determines whether P is in the interior of G.

Hint 1: Let P be any point not lying on any edge of G. Take any *ray* having P as its endpoint and count the number of edges of G that the ray intersects. If this number is odd, P is in the interior of G; otherwise it is not.[1] Intuitively, this makes sense: each time you "pass through" an edge, you are either passing from the polygon's interior to its exterior, or vice versa.

In practice, it is simplest to consider a ray that is either horizontal or vertical. The figure below illustrates the idea using "rightward-directed" horizontal rays. Each one is labeled by a number indicating how many edges of the polygon it intersects. Notice that the endpoints of the rays labeled with odd numbers are precisely those in the interior of the polygon.

[1] There are a few exceptional cases that will be discussed shortly.

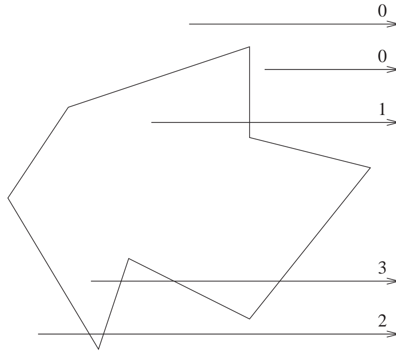

Figure 5: Counting Intersections of Rays and Edges

When counting how many edges of a polygon a ray intersects, we must be careful to handle two special cases. The first one occurs when the ray intersects two adjacent edges at their common endpoint. There are two sub-cases, each of which is depicted in the figure below. On the left side of the figure, the ray intersects edges $e1$ and $e2$ at their common endpoint. But because the other endpoints of $e1$ and $e2$ lie on *opposite sides* of the ray, we must treat this as though the ray intersects only one of the two edges rather than both of them.

On the right side of the figure, the ray intersects $e3$ and $e4$ at their common endpoint. But because the other endpoints of $e3$ and $e4$ lie on the *same side* of the ray, we treat this in the usual way by recording that the ray intersects both edges.

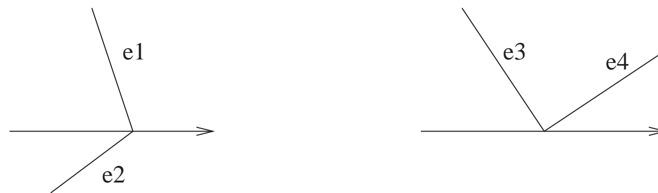

Figure 6: Special Case: Ray Intersects a Vertex

The second special case occurs when the ray wholly includes one of the polygon's edges. There are two sub-cases, each of which is depicted in the figure below. On the left side of the figure, the ray wholly includes $e2$, which also means that it intersects $e1$ and $e3$ at their respective common endpoints with $e2$. Because the other endpoints of $e1$ and $e3$ lie on *opposite sides* of the ray, we treat this in the usual way by recording that the ray intersects all three edges.

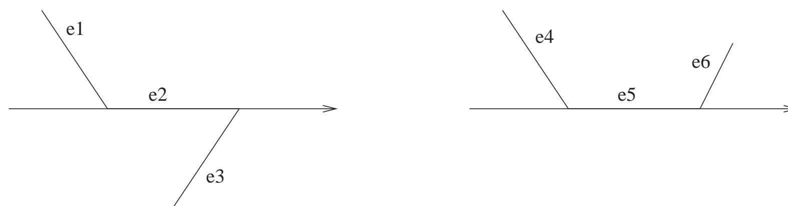

Figure 7: Special Case: Ray Includes an Edge

On the right side of the figure, the ray wholly includes $e5$, which also means that it intersects $e4$ and $e6$ at their respective common endpoints with $e5$. But because the other endpoints of $e4$ and $e6$ lie on the *same side* of the ray, we must treat this as though the ray intersects only two of the three edges.

Hint 2: Let $p_i = (x_i, y_i)$, $i = 1, 2$, and let L be the line $\overleftrightarrow{p_1 p_2}$ passing through p_1 and p_2. Assuming that $y_1 \neq y_2$ (so that L is not horizontal), the point at which L intersects the horizontal line given by $y = c$ is $(\alpha x_1 + (1 - \alpha)x_2,\ c)$, where $\alpha = \left|\frac{y_2 - c}{y_2 - y_1}\right|$.

Hint 3: The rightward-directed horizontal ray with endpoint (u, v) intersects the line segment connecting points p_1 and p_2 (as defined in Hint 2) only if $\min(y_1, y_2) \leq v \leq \max(y_1, y_2)$ and $u \leq \max(x_1, x_2)$.

End of Hints

Input: The input data describes a polygon (as a sequence of vertices), followed by some number of points, each of which is to be tested for being in the interior of the polygon. The first line contains a positive integer $m \geq 3$ indicating the number of vertices in the polygon. The next m lines describe these vertices, one per line, each given by a pair of real numbers indicating its x- and y-coordinate, respectively. On the following line is a positive integer n indicating how many points are to be tested. The next n lines describe these points, one per line, as with the vertices. You may assume that none of these points lies on any edge of the polygon.

Note that the sample data below corresponds to the polygon and the ray endpoints shown in the first two figures.

Output: With respect to the polygon given as input, for each point that is given as input, the program should identify it and report whether or not it is in the interior of the polygon.

```
Sample input                    Corresponding output
------------                    --------------------
8                               (2.0,7.75): non-interior
-1.0 -3.0                       (4.5,6.25): non-interior
0.0 0.0                         (0.75,4.5): interior
4.0 -2.0                        (-1.25,-0.75): interior
8.0 3.0                         (-3.0,-2.5): non-interior
4.0 4.0
4.0 7.0
-2.0 5.0
-4.0 2.0
5
2.0 7.75
4.5 6.25
0.75 4.5
-1.25 -0.75
-3.0 -2.5
```

Problem 7: Point in Interior of Polygon? — Again

In the previous problem, you were given the task of developing a program that, given a polygon and a set of points, determines, for each point, whether or not it lies in the interior of the polygon.

The hints given in that problem suggest that, to determine whether a point P lies in the interior of a polygon G, we take the rightward-directed horizontal ray having P as its endpoint and count how many edges of G the ray intersects. (An odd number indicates that P lies in the interior; an even number indicates otherwise.)

In the previous problem, however, you were told that you could assume that none of the given points would have the same y-coordinate as any of the vertices of the polygon. The purpose of this restriction was to ensure that, for each given point, the rightward-directed horizontal ray emanating from it would not pass through any vertex of the polygon.

In this problem, we lift that restriction. This makes things a bit more difficult because, in the case of a ray that passes through one or more vertices of the polygon, the *odd-implies-interior* and *even-implies-exterior* rule no longer applies, unless we modify the manner in which we count ray/edge intersections.

Specifically, when counting how many edges of a polygon a ray intersects, we must be careful to handle two special cases. The first one occurs when the ray intersects two adjacent edges at their common endpoint. There are two sub-cases, each of which is depicted in the figure below. On the left side of the figure, the ray intersects edges $e1$ and $e2$ at their common endpoint. Because the other endpoints of $e1$ and $e2$ lie on *opposite sides* of the ray, we must treat this as though the ray intersects only one of the two edges rather than both of them.

On the right side of the figure, the ray intersects $e3$ and $e4$ at their common endpoint. But because the other endpoints of $e3$ and $e4$ lie on the *same side* of the ray, we treat this in the usual way by recording that the ray intersects both edges.

Figure 8: Special Case: Ray Intersects a Vertex

The second special case occurs when the ray wholly includes one of the polygon's edges. There are two sub-cases, each of which is depicted in the figure below. On the left side of the figure, the ray wholly includes $e2$, which also means that it intersects $e1$ and $e3$ at their respective

common endpoints with $e2$. Because the other endpoints of $e1$ and $e3$ lie on *opposite sides* of the ray, we treat this in the usual way by recording that the ray intersects all three edges.

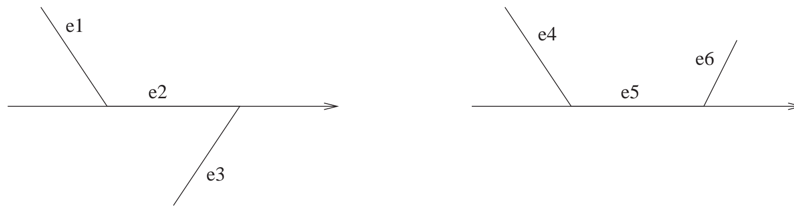

Figure 9: Special Case: Ray Includes an Edge

On the right side of the figure, the ray wholly includes $e5$, which also means that it intersects $e4$ and $e6$ at their respective common endpoints with $e5$. But because the other endpoints of $e4$ and $e6$ lie on the *same side* of the ray, we must treat this as though the ray intersects only two of the three edges.

The figure below illustrates the modified ray/edge intersection counting approach described above.

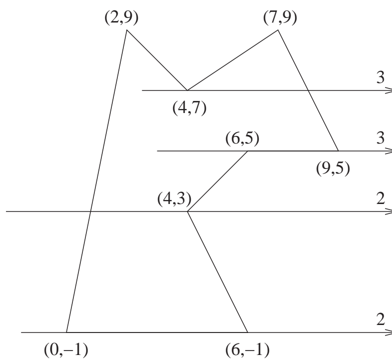

Figure 10: Modified Counting of Ray/Edge Intersections

Input: The input data describes a polygon (as a sequence of vertices), followed by some number of points, each of which is to be tested for being in the interior of the polygon. The first line contains a positive integer $m \geq 3$ indicating the number of vertices in the polygon. The next m lines describe these vertices, one per line, each given by a pair of real numbers indicating its x- and y-coordinate, respectively. On the following line is a positive integer n indicating how many points are to be tested. The next n lines describe these points, one per line, as with the vertices. You may assume that none of these points lies on any edge of the polygon.

Note that the sample data below corresponds to the polygon and the ray endpoints shown in the figure above.

Output: For each point that is given as input, the program should identify it and report whether or not it is in the interior of the polygon.

```
Sample input
------------
8
4.0 7.0
2.0 9.0
0.0 -1.0
6.0 -1.0
4.0 3.0
6.0 5.0
9.0 5.0
7.0 9.0
4
2.5 7.0
3.0 5.0
-2.0 3.0
-1.5 -1.0
```

```
Corresponding output
--------------------
(2.5,7.0): interior
(3.0,5.0): interior
(-2.0,3.0): non-interior
(-1.5,-1.0): non-interior
```