



Arbitrage?

`arbitrage.c`, `arbitrage.C`, `arbitrage.java`, `arbitrage.p`

If you are going to travel to the World Finals, you cannot rely on Czech Crowns. You would have to exchange your money for various foreign currencies. This problem deals with multiple currencies and their exchange rates. Your task is to verify that some set of exchange rates is safe, namely detect a possibility of so-called arbitrage.

An *arbitrage** is a risk-free combination of buy and sell operations that gains profit from imbalance in market prices. The prices may apply to various things, typically stock exchange but also currencies.

Input Specification

The input consists of several test cases. Each case begins with a line containing one positive integer number C , $1 \leq C \leq 200$, the number of currencies.

The second line of each test case contains C currency codes separated by a space. Each code is composed of 3 uppercase letters and all codes in one test case are different.

The third line contains one integer number R , $0 \leq R \leq C \cdot (C - 1)$, the number of exchange rates available. Each of the following R lines contains one exchange rate in the following format: first currency code, space, second currency code, space, integer number A_i , colon (":"), and integer number B_i . The meaning is as follows: If you pay A_i units of the first currency, you will get B_i units of the second currency. You may assume that $1 \leq A_i, B_i \leq 100$ and that the two currencies are different.

Output Specification

For each test case, print one line of output. If there exists any possible sequence of currency exchange operations that would result in a profit, the line should contain the word "Arbitrage". Otherwise, simply print "Ok".

The word *profit* in this case means that you start with any amount of any currency and after performing any number of exchanges you will have strictly higher amount of the same currency.

*Be careful and do not confuse "arbitrage" with "arbitration", which is something completely different; although the Czech language uses the same word for both.

Sample Input

```
2
CZK EUR
2
CZK EUR 25:1
EUR CZK 1:25
2
GBP USD
2
USD GBP 8:5
GBP USD 5:9
3
BON DEM CZK
3
DEM BON 1:6
BON CZK 1:5
DEM CZK 1:20
3
CZK EUR GBP
3
CZK EUR 24:1
EUR GBP 5:4
GBP CZK 1:30
3
CZK USD GBP
4
CZK USD 28:1
CZK GBP 31:1
GBP CZK 1:31
USD GBP 1:1
0
```

Output for Sample Input

```
Ok
Arbitrage
Ok
Ok
Arbitrage
```



CTU Open Contest 2010

Bus Schedules

`bus.c`, `bus.C`, `bus.java`, `bus.p`

Imagine that you happen to be the one to advance to the World Finals. Sounds good, doesn't it? Then you would be going to travel by various means of transport: airplanes, trains, buses, etc. Are you ready for that? This problem tries to evaluate your orientation skills in bus schedules ("time-tables").

As you know, not all buses operate on every-day basis. In bus schedules, the days when the bus operates are typically given by a list of various specifiers, some of them being simple ("only on Mondays"), some a little bit more confusing ("only on workdays"), and some totally impenetrable ("every day directly preceding Sunday or holiday, unless it is itself a holiday or a day following a holiday inside a leap year").

For the purposes of this problem, the specifier may be one the following:

"1"	The bus operates on Mondays.
"2" ...	The bus operates on Tuesdays, etc.
"7"	The bus operates on Sundays.
"t"	The bus operates on Sundays and official holidays, that is January 1, Easter Monday, May 1, May 8, July 5, July 6, September 28, October 28, November 17, and December 24, 25, and 26.
"w"	The bus operates on workdays, that is Monday to Friday except for the days specified in <code>t</code> .
"a"	The bus operates on workdays immediately following the days specified in <code>t</code> .

Furthermore, the days may be restricted only to specific dates or a range of dates, given as a comma-separated list of elements. Each element may be either a single date (written in the form of "`D.M.`", where `D` is the day number and `M` is the month number) or a date range (written as "`D1.M1.-D2.M2.`", where the first of the dates must precede the second one, and both of these boundary dates are *included* in the range). For example

3a 1.1.-30.6.,31.7.,1.9.-31.12.

describes a bus that goes only on Wednesdays and on workdays immediately following holidays, but only if one of these occurs on July 31 or between January 1 and July 30, or between September 1 and December 31 (inclusive).

Your task is to count all days that the bus operates in a given year.

Remarks:

- Today is Saturday, October 23, 2010.
- Each year has 365 days, except for leap years, in which an additional day (February 29) is inserted. A year Y is a leap year, if the number Y is evenly divisible by 4. The exception are years divisible by 100, which are leap years only if they are also divisible by 400.

In all non-leap years, if 29.2. is used as a single date, it is ignored; if it is used as a date-range start, it is interpreted as March 1, and if it is used as a date-range end, it is interpreted as February 28.

- Easter Monday is the Monday immediately following the Easter Sunday. Easter Sunday is the first Sunday *strictly after* the Paschal Full Moon date of the year. The Paschal Full Moon date for year y can be determined by the following algorithm:

```
int golden, solar, lunar, p;

golden = (y % 19) + 1;
solar = (y - 1600) / 100 - (y - 1600) / 400 ;
lunar = (((y - 1400) / 100) * 8) / 25;
p = (3003 - (11 * golden) + solar - lunar) % 30;
if (p == 29 || (p == 28 && golden > 11)) p--;
```

Paschal Full Moon date is then p days ($0 \leq p \leq 28$) after March 21.

Input Specification

The input contains several test cases, each case consists of a single line. The line contains two non-empty strings S , R , and an integer Y ($1600 \leq Y \leq 3000$) separated by a space. The string S will contain only characters “1”, ..., “7”, “t”, “w”, “a”, each appearing at most once (in any order). The string R of length at most 1000 is a comma-separated list of dates and date ranges whose format was specified above. The dates and date ranges in R do not overlap and are always given in an ascending order.

Output Specification

For each input instance, output a single line containing one integer, the number of days in the year Y that belong to the range R and satisfying at least one condition specifier given by S .

Sample Input

```
3a 1.1.-30.6.,31.7.,1.9.-31.12. 2010
```

Output for Sample Input



Bus Clock Display

`clock.c`, `clock.C`, `clock.java`, `clock.p`

When you travel to Egypt, you will probably want to enjoy the famous pyramids and possibly also to take a bus to Luxor to visit the Valley of the Kings. To overcome boredom during such a long trip, imagine you are trying to keep yourself occupied by observing a large clock above the driver's seat. The clock is a digital one and uses the classical 7-segment display. Since the Egyptian buses are crowded and most local people use to carry strange things with them, your view may be obstructed and some segments possibly hidden.

If you think about such situations, you will quickly notice that it is sometimes possible to read the time reliably even if some of the segments are not visible. For example, in Figure 1, the clock cannot show anything else than 12:04.

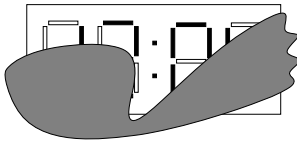


Figure 1

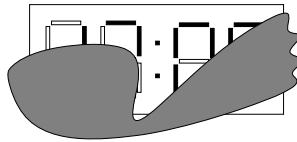


Figure 2

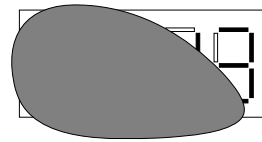


Figure 3

In some other cases, there may be several possibilities and you cannot tell for sure what time is displayed. Figure 2 is a good example – the last digit may be 0, 5, 6, 8, or 9. But even if you cannot tell the time immediately, you do not lose hope. After a while, you take another look at the clock and see another “sample” of it. Meanwhile, the people moved a little bit, so you can see a different portion of the display now, as shown in Figure 3. The new picture is even more ambiguous if seen alone. But you are quite sure that no more than four minutes passed between the two samples. With this knowledge, you may not only say what time it is now, but also retroactively determine what time it was before.

Your task is to write a program that automates this process. Given samples of the display (with some segments potentially hidden) and the estimate of the time elapsed between every two consecutive samples, you have to compute what time was displayed at the clock when the samples were taken.

Input Specification

The input contains several bus descriptions. Each description starts with a positive integer number S , which denotes the number of samples taken, $1 \leq S \leq 100$. Then the individual samples follow. Each sample is represented by 28 characters that give states of all segments, in the order shown in Figure 4. The characters are separated by at least one space or new-line. Additional spaces may be used for better presentation in printed materials.

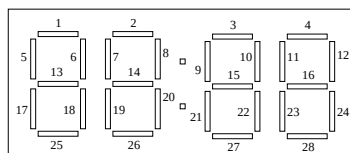


Figure 4

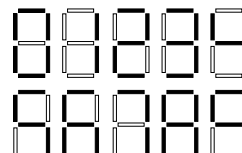


Figure 5

Each character is either a dash “-” when you see that a horizontal segment is lit, a pipe character “|” for a vertical segment lit, a dot “.” for a horizontal or vertical segment that you know is not lit, or a question mark “?” if the corresponding segment cannot be seen at all.

Shapes of individual digits used by the display are shown in Figure 5. The clock uses the 24-hour format with values between 0:00 and 23:59. The first of four digit positions never displays zero, it will be blank if the number of hours is less than 10.

Between each two consecutive samples, there are two integer numbers separated by a space, T_{min} and T_{max} , $0 \leq T_{min} \leq T_{max} \leq 120$, denoting the minimal and maximal number of minutes that may have elapsed between those two samples. The last bus description is followed by a line containing zero.

Output Specification

For each bus description, print one line for every sample taken in that bus. If the time shown by the clock can be determined uniquely, print the time as three or four digits with a colon between hours and minutes. If the time is ambiguous, print the sentence “ambiguous, P_i possibilities”, where P_i is the total number of different times that might have been displayed at that moment. Remember to always consider all those (and only those) possibilities that are consistent with *all* samples. Print one empty line after each bus.

Sample Input

```

3
- - . -
. | . | | |
- - - .
| . . | . | |
- - . -
16 30
? ? ? -
? ? ? ? ? ? | .
? ? ? ? -
? ? ? ? ? ? . |
? ? ? ? -
15 30
? ? ? ?
? ? ? ? ? ? ? |
? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ?
1
. - - .
. | . | | | ?
? ? ? ?
? ? ? . | ? ? ?
? ? ? ?
2
. - - -
. | . | | | ?
? ? ? ?
? ? ? . | ? ? ?
? ? ? ?
0 4
? ? . -
? ? ? ? ? | . |
? ? ? -
? ? ? ? ? ? ? |
? ? ? -
0
```

Output for Sample Input

```

23:40
0:05
ambiguous, 13 possibilities

12:04

12:09
12:13
```



CTU Open Contest 2010

Measuring Problem Difficulty

`difficult.c`, `difficult.C`, `difficult.java`, `difficult.p`

If you are the lucky one to advance to the ACM-ICPC World Finals, one of the situations you will face is the world finals competition itself. Wait, isn't that the *main* reason to go there?

In the beginning of each ACM-ICPC competition, there are two separate goals each team tries to accomplish:

1. among the given set of problems, find the easiest one,
2. solve that problem as fast as possible.

To evaluate the performance of all teams in detail, we want to test your abilities for each of these two goals separately. This problem deals with the former goal (finding the easiest problem), the latter one (solving it) is analyzed in another problem (*easy*).

The main trouble with comparing problem difficulty is that the opinions of different people may vary. To satisfy everyone, we need to find some consensus. Let's start with determining all problems on which the opinions agree.

Your team is given a set of ICPC problems. Each team member sorts all of the problems in the order of their expected difficulty. Then we want to find all pairs of problems such that their relative order is the same according to all given orderings.

Input Specification

The input contains several tasks. Each task starts by one line containing single integer N , $2 \leq N \leq 150\,000$, the number of problems to consider.

After that, there are three blocks, each of them describing opinion of one team member. (ICPC teams have three members, right?) Every block specifies an arbitrary permutation of N numbers $1 \dots N$ representing the problems. You should know from the school that the word "permutation" means each of the numbers will appear exactly once in each block.

Each block starts on a new line. For presentation reasons, the numbers inside a block may be split among any number of lines. If there are more than one number per line, they will be separated by at least one space. Empty lines may occur before and after blocks.

The last task is followed by a line containing zero.

Output Specification

For each task, print one line containing a single integer number: the number of all pairs of problems, whose mutual order is the same in all three permutations.

Please note that the result may be as high as $\frac{N \cdot (N-1)}{2}$ and may therefore exceed 2^{32} .

Sample Input

4

1 2 3 4

2 3 4 1

4 1 2 3

6

3 6 1 2 4 5

3 6 1 4 2 5

3 6 1 2 4 5

0

Output for Sample Input

1

14



Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina

Czech Technical University in Prague

Technical University of Ostrava
Masaryk University
Pavol Jozef Šafárik University in Košice



CTU Open Contest 2010

The Easiest Problem is This One

`easy.c`, `easy.C`, `easy.java`, `easy.p`

If you are the lucky one to advance to the ACM-ICPC World Finals, one of the situations you will face is the world finals competition itself. Wait, isn't that the *main* reason to go there?

In the beginning of each ACM-ICPC competition, there are two separate goals each team tries to accomplish:

1. among the given set of problems, find the easiest one,
2. solve that problem as fast as possible.

To evaluate the performance of all teams in detail, we want to test your abilities for each of these two goals separately. The former goal (finding the easiest problem) is analyzed in another problem (`difficult`), here we deal with the latter goal (solving the easiest problem).

To isolate other influences, we will tell you clearly which problem is the easiest one to solve in this problem set (CTU Open Contest 2010). It is this one! What more can we do to emphasize that fact? The title says it. The problem name is `easy`. And believe us, it is true. This is indeed the easiest of all nine problems. We recommend you to do it first.

A positive integer number N can be expressed in the decimal system (numeral system with the base of 10) as the sequence of digits d_i

$$N = d_1 d_2 d_3 d_4 \dots d_k$$

where $\forall i, 1 \leq i \leq k : 0 \leq d_i \leq 9$.

The digits express the value which has to be multiplied by a power of ten:

$$N = d_1 \cdot 10^{k-1} + d_2 \cdot 10^{k-2} + \dots + d_{k-1} \cdot 10 + d_k = \sum_{i=0}^{k-1} d_{i+1} \cdot 10^{k-i-1}$$

The sum of the digits $S(N)$ is then defined as the plain sum of all individual digits without multiplying them by powers of ten:

$$S(N) = d_1 + d_2 + d_3 + \dots + d_k = \sum_{i=0}^{k-1} d_{i+1}$$

For example:

$$\begin{aligned} N &= 3029 = 3 \cdot 10^3 + 2 \cdot 10 + 9 \\ S(N) &= 3 + 0 + 2 + 9 = 14 \end{aligned}$$

If we multiply the original number N with another number m , the sum of the digits typically changes. For example, if $m_1 = 26$:

$$N \cdot m_1 = 78754 = 7 \cdot 10^4 + 8 \cdot 10^3 + 7 \cdot 10^2 + 5 \cdot 10 + 4$$

$$S(N \cdot m_1) = 7 + 8 + 7 + 5 + 4 = 31$$

However, there are some numbers that, if multiplied by N , will result in the same sum of the digits as the original number N . For example, consider $m_2 = 37$:

$$N \cdot m_2 = 112073 = 10^5 + 10^4 + 2 \cdot 10^3 + 7 \cdot 10 + 3$$

$$S(N \cdot m_2) = 1 + 1 + 2 + 0 + 7 + 3 = 14 = S(N)$$

Your task is to find the smallest positive integer p among those that will result in the same sum of the digits when multiplied by N . To make the task a little bit more challenging, the number must also be higher than ten.

Input Specification

The input consists of several test cases. Each case is described by a single line containing one positive integer number N , $1 \leq N \leq 100\,000$. The last test case is followed by a line containing zero.

Output Specification

For each test case, print one line with a single integer number p which satisfies *all* of the following conditions:

- $p > 10$
- $S(N) = S(N \cdot p)$
- $\forall q \in \mathbb{N} : [S(N) = S(N \cdot q)] \Rightarrow [(q \geq p) \vee (q \leq 10)]$

Sample Input

```
3029
4
5
42
0
```

Output for Sample Input

```
37
28
28
25
```

CTU Open Contest 2010

Prohibición de fumar

`fumar.c`, `fumar.C`, `fumar.java`, `fumar.p`

When travelling abroad (to the World Finals, for example), you should always obey all local laws of every country you visit. In some countries, they have laws that prohibit smoking closer than some given distance D from any building. In a modern world, it becomes a pressing question whether there is any place where you could legally smoke in a town at all.*

Fortunately, the town we consider here was built in an orderly way, making it easier to decide this question. The boundary of the town is an axis-parallel rectangle R and the buildings are pairwise disjoint axis-parallel rectangles contained inside R . Since police patrols are generally not equipped with micrometers, it is possible to smoke at any spot within the town (including the boundary rectangle) whose distance to the closest building is at least $(D - 1)$ meters and 90 centimeters.

Input Specification

The input contains several towns. Each town description consists of several lines. The first line contains four integers D , R_x , R_y , and N separated by a space. The number D ($1 \leq D \leq 100\,000$) gives the minimum distance from the buildings at which the smoking is legal. R_x and R_y ($1 \leq R_x, R_y \leq 100\,000$) give the dimensions of the town, which covers a rectangle with vertices at the coordinates $(0, 0)$, $(R_x, 0)$, (R_x, R_y) , and $(0, R_y)$. Finally, N ($0 \leq N \leq 200$) is the number of buildings in the town.

Each of the following N lines contains four integers F_x , F_y , T_x , T_y ($0 \leq F_x < T_x \leq R_x$, $0 \leq F_y < T_y \leq R_y$), giving the coordinates (F_x, F_y) , (F_x, T_y) , (T_x, T_y) , and (T_x, F_y) of the corners of the particular buildings.

The last town is followed by a line containing four zeros.

Output Specification

For each input instance, if there is any place in the town where smoking is permitted, output a single line containing two real numbers X and Y rounded to two decimal places, giving the coordinates (X, Y) of *any* point inside the town boundaries whose distance is at least $D - 0.1$ m from any of the buildings. Otherwise, the line should contain the string “Smoking not permitted!”.

To avoid rounding errors, you may assume that the town is built in such a way that either there exists a point whose distance from every building is at least $D + 0.1$ m, or that all points in the town are closer than $D - 0.1$ m to some building.

*Or, from the other perspective, whether your favorite town finally became smoke-free.

Sample Input

```
8 20 20 1
7 7 13 13
10 20 20 1
7 7 13 13
0 0 0 0
```

Output for Sample Input

```
1.34 1.34
Smoking not permitted!
```



Pyramid Guards

`guard.c`, `guard.C`, `guard.java`, `guard.p`

Without a doubt, the Great Pyramid of Giza is the most famous tourist attraction in Egypt. It is a less known fact that building it was only a part of the task. The other part was security. When the Pyramid was finished, some people had an opinion that the Pharaoh's corpse does not really need all of the fancy treasures. So, they wanted to take something for themselves. To prevent the Pyramid from being robbed, a pair of guards were constantly walking around in opposite directions. To allow for better planning, your task is to determine the closest distance from which the guards can see each other in any moment.

The Great Pyramid is a pyramid (in the terms of geometry) with a square base and its apex perpendicularly above the center of the base. The original base length is believed to be 440 cubits* and the height is 280 cubits.

Both guards move on the surface of the pyramid, one of them walking clockwise and the other counter-clockwise around the pyramid apex. The path of each guard forms a quadrilateral – it has four line segments, each of them running through one side of the pyramid. After reaching his[†] starting point, the guard immediately continues with another round across the exactly same path.

Input Specification

The input consists of several test cases. Each test case is composed of two lines, each of them describing the path of one guard. Each line contains five integer numbers: H_1 , H_2 , H_3 , H_4 ($0 \leq H_i < 280$) and T ($1 \leq T \leq 120$).

The guard starts at one of the edges at height H_1 (measured vertically from the ground level). The other three numbers give heights at which he crosses the other three edges of the pyramid. T stands for the number of minutes after which the guard returns back and starts another loop.

At the beginning, both guards start at the same moment at opposite edges of the pyramid, walking in opposing directions. They always move at a constant speed and you may assume that their shift is infinitely long.[‡]

The last test case is followed by a line containing five zeros.

* *Cubit* was the unit of length used in the ancient Egypt.

[†] A modern and politically correct text should use either the word *their* or explicitly mention both variants: *his* and *her*. However, in the ancient Egypt, the people did not care about gender emancipation and women were simply not allowed to guard pyramids.

[‡] Well, several thousand years, at least.

Output Specification

For each test case, output the minimal distance between the two guards. Consider only such cases when the guards can really see each other, i.e., when they walk the same side/edge of the pyramid.

The distance must be printed in cubits, with exactly three digits after the decimal point.

Sample Input

```
0 0 0 0 20
0 0 0 0 17
100 200 100 200 50
50 150 50 150 60
0 0 0 0 0
```

Output for Sample Input

```
0.000
44.481
```



Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina

Czech Technical University in Prague

Technical University of Ostrava
Masaryk University
Pavol Jozef Šafárik University in Košice



CTU Open Contest 2010

Hack around the Lock

`hack.c`, `hack.C`, `hack.java`, `hack.p`

For your travel to the ACM-ICPC World Finals in Egypt, you have bought a brand new luggage. The luggage has a numeric lock composed of K wheels with digits $0 \dots 9$ on each of them. Each combination of the values on the wheels can be represented as a K -digit number (possibly with leading zeros that count as digits). Only one combination opens the luggage.

As a little bit paranoid programmer, you wrote a sophisticated random number generator to choose the secret number and you can feel safe now because nobody will ever be able to guess it easily.

Unfortunately, after arriving to the hotel, you realize “nobody” does also include you. Sad story, isn’t it? Instead of starting to frenetically rotate the wheels trying to find the right combination, you decided to write another program that will help you. Given an initial numeric combination, the program should find fastest way to try all other possible combinations — as we all know, the correct combination will always be the one we try at the very last.

It is possible to rotate one wheel at a time only. After changing the digit by one, you can check if the combination is correct. You also know that the initial combination is *not* correct (otherwise, you wouldn’t need the whole thing at all, would you?). So, the overall time needed to open the luggage is proportional to the number of *steps*, each step corresponding to the rotation of some wheel by one. To make the things worse, it is impossible to turn a wheel directly from 0 to 9 or vice versa — you need 9 steps instead.

Input Specification

The input contains several instances; input of each instance is a line with a single decimal number N , which is the initial combination. The number may have leading zeros, which are counted into its number of digits K , $1 \leq K \leq 7$. For example, “007” is considered to be a 3-digit number. The last instance is followed by a line containing “-1”.

Output Specification

For each instance, print two lines: the first one containing the minimum possible number of steps S . The second line must contain a sequence of S K -digit numbers. Every possible K -digit number (except for the initial one) must appear at least once in the sequence and each two consecutive numbers must differ in exactly one digit by exactly one (in the digit value).

Sample Input

5
00
-1

Output for Sample Input

13
6 7 8 9 8 7 6 5 4 3 2 1 0
99
10 20 30 40 50 60 70 80 90 91 81 71 61 51 41 31 21 11 01
02 12 22 32 42 52 62 72 82 92 93 83 73 63 53 43 33 23 13 03
04 14 24 34 44 54 64 74 84 94 95 85 75 65 55 45 35 25 15 05
06 16 26 36 46 56 66 76 86 96 97 87 77 67 57 47 37 27 17 07
08 18 28 38 48 58 68 78 88 98 99 89 79 69 59 49 39 29 19 09

The last line of the sample output above is cut into several lines to fit into the page width, but it should be a single line in the real output.



CTU Open Contest 2010

God Save the i -th Queen

`ith.c, ith.C, ith.java, ith.p`

Did you know that during the ACM-ICPC World Finals a big chessboard is installed every year and is available for the participants to play against each other? In this problem, we will test your basic chess-playing abilities to verify that you would not make a fool of yourself if you advance to the World Finals.

During the yesterday's Practice Session, you tried to solve the problem of N independent rooks. This time, let's concentrate on queens. As you probably know, the queens may move not only horizontally and vertically, but also diagonally.

You are given a chessboard with $i - 1$ queens already placed and your task is to find all squares that may be used to place the i -th queen such that it cannot be captured by any of the others.

Input Specification

The input consists of several tasks. Each task begins with a line containing three integer numbers separated by a space: X, Y, N . X and Y give the chessboard size, $1 \leq X, Y \leq 20\,000$. $N = i - 1$ is the number of queens already placed, $0 \leq N \leq X \cdot Y$.

After the first line, there are N lines, each containing two numbers x_k, y_k separated by a space. They give the position of the k -th queen, $1 \leq x_k \leq X, 1 \leq y_k \leq Y$. You may assume that those positions are distinct, i.e., no two queens share the same square.

The last task is followed by a line containing three zeros.

Output Specification

For each task, output one line containing a single integer number: the number of squares which are not occupied and do not lie on the same row, column, or diagonal as any of the existing queens.

Sample Input

```
8 8 2
4 5
5 5
0 0 0
```

Output for Sample Input

20