A

acm 2006 Greater New York Programming Contest
Nassau Community College
Garden City, NY

event sponsors IBM AdaCore The GNAT Pro Company Google

E • Sign Message Formatting

The *National Transportation Communications for ITS Protocol* (NTCIP) for communicating with highway signs with dynamic messages describes the messages using *Markup Language for Transportation Information* (MULTI). A MULTI string consists of text to be displayed together with embedded tags which describe formatting of the text and included dynamic elements. Tags begin with the open bracket ('[') character and end with the close bracket (']') character. If an open bracket character is to appear in the text, it is represented as two open bracket characters. Similarly, if a close bracket character is to appear in the text, it is represented as two close bracket characters. This problem is concerned with formatting for character cell displays, which are rectangular arrays of character cells each of which can display a single character.

The tags to be supported for this problem are:

Tag	Description
`[nl]`	Start a new row of the array
`[sc<digit>]`	Insert <*digit*> blank character cells between each pair of text characters in the following string until changed.
`[/sc]`	Set inter-character spacing to zero (equivalent to `[sc0]`).
`[jl2]`	Set left justified text (the first character of the text is the leftmost character of the line).
`[jl3]`	Set center justified text (the number of character positions before and after the text on the line is the same or the number after is one more than the number before).
`[jl4]`	Set right justified text (the final character of the string is the rightmost character on the line).
`[jl5]`	Set fully justified text (an equal number of blank character spaces, as large as possible for the line length, is placed between each pair of characters in the text; the resulting string is centered in the line as for centered text).

For example, on a 24 character line (□ indicates a blank character cell):

Format String	Generated Output
`[jl2]MESSAGE`	`MESSAGE□□□□□□□□□□□□□□□□□`
`[jl3]MESSAGE`	`□□□□□□□□MESSAGE□□□□□□□□`
`[jl4]MESSAGE`	`□□□□□□□□□□□□□□□□□MESSAGE`
`[jl2][sc2]MESSAGE`	`M□□E□□S□□S□□A□□G□□E□□□□□`
`[jl5]MESSAGE`	`□□M□□E□□S□□S□□A□□G□□E□□□`
`[jl2]THIS[jl3]IS A[jl4]MESSAGE`	`THIS□□□□□□IS A□□□MESSAGE`

Letters within tags are case-insensitive. That is `[nl]` = `[NL]` = `[Nl]` = `[nL]`.

The default justification at the beginning of a message is left justification and the default character spacing is 0.

Justification and character spacing are maintained across `[nl]` tags.

The `[jl2]` and `[jl5]` tags may only be used before any text has been output on a line. Otherwise it is an error (**TAG CONFLICT)**.

Once `[jl5]` text has been output on a line, no other justification tag may be set on that line. Otherwise it is an error (**TAG CONFLICT)**.

The `[jl3]` tag may not be used after right justified text (`[jl4]`) has been output on a line. Otherwise it is an error (**TAG CONFLICT)**.

A justification tag **[jl?]** with the same value as the current value does not cause a **TAG CONFLICT** error.

Extra character spacing specified by the **[sc?]** tag is ignored on lines with fully justified text. The full justification rules determine the extra spaces.

If too many characters are required on a line or too many lines are required in a message, it is an error (**TOO BIG**). A **[nl]** tag does not begin a new line unless followed by text output.

If left justified text and center justified text appear on the same line, there must be at least one blank character cell between the last character of left justified text and the first character of center justified text. Otherwise it is an error (**TOO BIG**).

If center justified text and right justified text appear on the same line, there must be at least one blank character cell between the last character of center justified text and the first character of right justified text. Otherwise it is an error (**TOO BIG**).

If left justified text and right justified text appear on the same line, there must be at least one blank character cell between the last character of left justified text and the first character of right justified text. Otherwise it is an error (**TOO BIG**).

The only tags allowed in a message are the seven tags listed above otherwise it is an error (**BAD TAG**). A malformed tag or an unmatched single open or closed bracket is a **BAD TAG** error.

For this problem you will write a program which takes as input the dimensions of the character cell array and a MULTI string and either outputs an error string or a correctly formatted message.

Input

The first line of input contains a single integer ***N***, (1 ≤ ***N*** ≤ 100), which is the number of datasets that follow. Each dataset consists of a single line containing an integer ***R***, (1 ≤ ***R*** ≤ 25), a blank, an integer ***C***, (1 ≤ ***C*** ≤ 80), a blank, and the remainder of the line is a ***MULTI-Text*** string. ***R*** is the number of rows in the character cell array, ***C*** is the number of columns in the character cell array, and the ***MULTI-Text*** is the text to be formatted.

Output

For each dataset, output the dataset number on a line by itself, followed by one of the error strings (TAG CONFLICT, TOO BIG, BAD TAG) on a line by itself in the case of an error, or, ***R*** lines each of which has exactly ***C*** characters (other than terminating newlines) representing the formatted message using space characters for empty character cells. The last line of output for a dataset result should be a single blank line.

Note: For ease in grading, a dataset that contains an error will only contain one type of error.

Sample Input

```
7
4 24 [jl2]MESSAGE[nl][jl3]MESSAGE[nl][jl4]MESSAGE
2 24 This[jl3]is a[jl4]message
2 24 This is a very long message which will not fit
4 24 This[nl]message[nl]has[nl]too[nl]many[nl]lines
2 32 [jl3]This message has a [[ and a ]]
2 32 This is a bad tag[xy34]
2 32 [jl3]This message [jl5] has a tag conflict.
```

Sample Output
(^ added after the last character of the lines for illustrative purposes. It should NOT appear in program output.)

```
1
MESSAGE                 ^
        MESSAGE         ^
                 MESSAGE^
                        ^

2
This      is a   message^
                        ^

3
TOO BIG

4
TOO BIG

5
  This message has a [ and a ]  ^
                                ^

6
BAD TAG

7
TAG CONFLICT
```

C: Museum Guards

A museum has hired some guards and is trying to build a schedule for them. The museum would like to build a 24-hour schedule for the guards, such that:

- Each guard works during the same time intervals every day.
- Each guard works within his/her time windows of availability, which s/he specifies.
- Each guard works at most the amount of time s/he is able, which s/he also specifies.
- The guards can only shift (start or stop working) on half hour boundaries. (e.g. 04:00 or 04:30, but not 04:15).
- The guards are only scheduled for shifts if they are available to work at all times during those shifts. (e.g. if a guard's window of availability opens at 03:05, they cannot be scheduled at 03:00.)
- The minimum number of guards on duty at any time during the day is maximized. This improves security of the museum.

Write a program to help the museum staff determine the maximum number of guards that they can maintain at all times throughout a 24-hour day, given the constraints of the guards' availability. You may assume that guard exchanges are instantaneous. That is, if 2 guards leave and 2 other guards arrive at the same time, the museum is guarded by 2 guards through this exchange.

Input

There will be multiple test cases. Each test case begins with a line containing a single integer **`N`** (**`1`** ≤ **`N`** ≤ **`50`**), the number of guards available. There will then be **`N`** blocks of data, one for each guard.

Each block provides the preferences of one guard. A block begins with two integers, **`K`** (**`1`** ≤ **`K`** ≤ **`50`**) and **`M`** (**`1`** ≤ **`M`** ≤ **`1440`**), in that order; **`K`** is the number of time intervals specifying when the guard is available for work, and **`M`** is the maximum number of minutes s/he is able to work each day. The next **`K`** lines each contains the starting and ending time, in that order, of a time interval where the guard is available, separated by whitespace. These time intervals may overlap. The *union* of all **`K`** time intervals provides the complete set of times at which the guard is available.

A starting or ending time is formatted as **`HH:MM`** (`00` ≤ **`HH`** ≤ `23`, `00` ≤ **`MM`** ≤ `59`). Midnight is represented by `00:00`. When the ending time is smaller than the starting time, it means the guard is available for working past midnight. For example, the interval "`23:00 03:00`" means the guard is available from 11pm at night to 3am in the morning. If the starting and ending times are equal, then the guard is available for work during any time intervals throughout the day. The last test case is followed by a line with a single `0`.

Output

For each test case, output a single integer, representing the minimum number of guards on duty at any given time during the day, using a schedule that maximizes this value. That is, output the largest integer k, such that there is a schedule where at any given moment, there are k guards on duty at the museum, assuming instantaneous exchanges specified above. Do not print any blank lines between answers.

Sample Input

```
3
1 540
00:00 00:00
3 480
08:00 10:00
09:00 12:00
13:00 19:00
1 420
17:00 00:00
5
1 720
18:00 12:00
1 1080
00:00 23:00
1 1080
00:00 20:00
1 1050
06:00 00:00
1 360
18:00 00:00
3
1 1440
00:00 00:00
1 720
00:00 12:15
1 720
12:05 00:15
0
```

Sample Output

```
1
2
1
```

Southeastern European Regional Programming Contest
Bucharest, Romania
October 17, 2009

Problem E
Multiprocessor Scheduling

Input File: E.IN
Output File: standard output
Program Source File: E.C, E.CPP, E.JAVA

There are two applications running on a multiprocessor machine. Each application i ($i=1,2$) consists of N procedures which are numbered from 1 to N and must be executed sequentially (in the order $1,…,N$). A procedure will be identified by a pair (i,j), where $i=1,2$ identifies the application and $1\leq j\leq N$ represents the index of the procedure in the sequence of procedures of the application i. A procedure (i,j) can only be executed on the processor $P(i,j)$ of the machine and its execution lasts for $D(i,j)$ seconds. We want to schedule the execution of the procedures of the two applications over the processors of the machine in such a way that the time moment when the last procedure finishes its execution (from any of the two applications) is minimum; this time moment is called *makespan*. We consider that the two applications are available for scheduling starting from the time moment 0. The schedule needs to obey the following rules:

- once the execution of a procedure (i,j) starts on the processor $P(i,j)$, we cannot interrupt it until the execution of the procedure ends
- we cannot execute multiple procedures on the same processor at the same time, but we can execute multiple procedures in parallel on different processors
- the execution of the procedure (i,j) ($2\leq j\leq N$) starts either at the exact time moment tm when the procedure $(i,j-1)$ finishes its execution or at any time moment after tm
- if a procedure begins its execution at time moment tm, then it will finish its execution at the time moment $tm+D(i,j)$

Write a program that, given the information regarding the procedures of the two applications, computes the minimum makespan.

The first line of the input file contains the number T of test cases which are described next. The first line of a test case contains the number N ($1\leq N\leq 300$) of procedures composing each of the two applications. Then, N lines follow, describing the first application. The j^{th} of these N lines contains two integers, separated by a blank: $P(1,j)$ and $D(1,j)$. After this, other N lines follow, describing the second application. The j^{th} of these N lines contains two integers, separated by a blank: $P(2,j)$ and $D(2,j)$. We have $1\leq P(i,j)\leq 10$ and $1\leq D(i,j)\leq 15000$ ($i=1,2$; $1\leq j\leq N$). Notice that we may have $P(i,j)=P(k,l)$ – this implies that the procedures (i,j) and (k,l) cannot be executed during overlapping time intervals (notice also that if $i=k$ this would not matter, as the procedures of the same application must be executed sequentially).

The output file must contain exactly T lines with a single number each – the minimum makespan for the corresponding test from the input file. These answers must be printed in the order in which the test cases are given in the input file (i.e. the i^{th} line of the output file contains the answer to the i^{th} test case from the input file). An input/output sample follows.

Input	Output

2
1
2 6
1 10
3
2 31
2 18
4 15
2 26
3 40
5 16
	10
	90

C

[H] Land Division

Program:	`land.(c|cpp|java)`
Input:	`land.in`
Balloon Color:	`Gold`

Description

The king of the *Far, Far Away Kingdom* has passed-away and the kingdom must be split amongst his K sons. The kingdom, which can be drawn on a rectangular map, consists of N cities. To divide the land, they will draw $K-1$ straight segments on the map, all of them parallel to either the vertical or the horizontal axis of the map. This divides the map into exactly K rectangles, all having equal heights (if the dividing lines where vertical), or equal widths (if the dividing lines where horizontal). No segment should pass through any of the cities. Each son will then be assigned one random region out of the K regions and the cities inside that region will be his share.

Of course, they want the division to be as fair as possible: theoretically, in the fairest division, each son should get N/K cities (we'll call this value the baseline), but since the baseline isn't always a whole number, each of the sons wants to be as close as possible to the baseline. We will calculate the unfairness of each son as the absolute difference between the number of cities assigned to him and the baseline. The fairest division is the one that minimizes the average unfairness of all the sons.

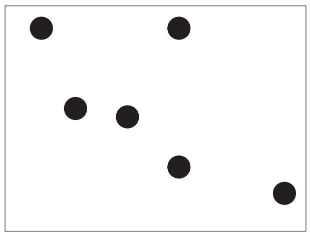


Figure (a)

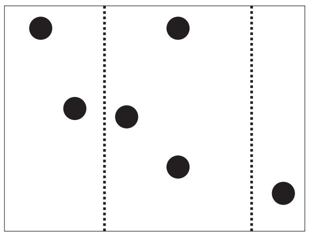


Figure (b)

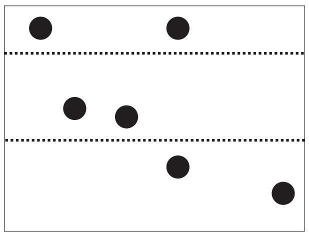


Figure (c)

Consider the example above with 3 sons and 6 cities (so the baseline is $6/3 = 2.0$) Figure (a) is the original map. Figure (b) shows a non-optimal division (the dashed lines are the 2 dividing lines.) In this case, the middle region contains 3 cities (unfairness of $|3-2| = 1$), the left region contains 1 city (unfairness of $|1-2| = 1$), while the right region contains 2 cities (perfectly fair, unfairness of 0), so the average unfairness is $2/3$.

Figure (c) on the right shows the optimal division since all three regions contain the same number of cities for an average unfairness of 0.

Write a program to determine the fairest land division for a given kingdom.

Input Format

Your program will be tested on one or more test cases. Each test case is described on $N+1$ lines. The first line of each test case specifies two positive integers: ($N \leq 100,000$) and ($K \leq 10$) where N is the number of cities and K is number of children. Note that $K \leq N$.

N lines follows, each describing the coordinates of a city by specifying two integers (x, y) where $0 \leq x, y \leq 100,000$. Since coordinates are rounded to the nearest integer, more than one city could have the exact same coordinate on the map. You may assume that the map of the kingdom is any rectangle that contains all of the given points (although such information is not needed by the program.) Note also that while all cities lie on integer coordinates, the dividing lines need not be.

The last line of the input file contains two zeros.

Output Format

For each test case, print the following line:

```
k.␣A/B
```

Where `k` is the test case number (starting at one,) and `A/B` is the minimum average that could be obtained. `A/B` should be an irreducible fraction. Let `B=1` when the result is a whole number.

Sample Input/Output

land.in

```
6 3
0 4
1 3
2 3
3 1
4 4
5 0
4 3
0 0
0 1
1 1
1 0
0 0
```

OUTPUT

```
1. 0/1
2. 8/9
```

acm 2008
Greater New York Programming Contest
St. Joseph's College
Patchogue, NY
event sponsors
IBM AdaCore The GNAT Pro Company

F · Extended Normal Order Sort

When sorted in standard order, strings with digits in them may not sort to where they are expected. For instance, **xyz100** precedes **xyz2**. In some applications such as listing files, *normal order sort* may be used where any string of digits in a character string is treated as a single digit with numerical value given by the digit string. For example, the following are in normal order:

XYZ001, **XYZ2**, **XYZ003**, **XYZ08**, **XYZ23**, **XYZ100**, **XYZQ**

We wish to extend normal order sort in two ways:

1. Lower case and upper case letters sort the same (with the upper case value).
2. If a plus (+) or minus (-) sign precedes a digit and does not follow a digit, it is considered part of the following number for sorting purposes.

So **123+456+7890** are three numbers separated by plus signs but **A+003** is the same as **A3.**

To do our sort, we will use a library sort routine but we need to furnish a comparison routine. Write a comparison routine which takes as input two strings of printable, non-space *ASCII* characters (**`chr(33)-chr(126)`**) and returns:

-1 if the first string should precede the second in extended normal order
0 it the two strings are the same in extended normal order, or
1 if the first string should follow the second in extended normal order.

Input

The first line of input contains a single integer N, ($1 \leq N \leq 1000$) which is the number of data sets that follow. Each data set consists of a single line of input containing the two strings to be compared separated by a space.

Output

For each data set, you should generate one line of output with the following values: The data set number as a decimal integer (start counting at one), a space and **–1**, **0** or **1** depending on whether the first string precedes, is the same as, or follows the second string in extended normal order.

Sample Input

```
5
x-3 X0001
123-456-7890 123+456+7890
xYz000123J XyZ+123j
#$%^&*[]- abcdefgh
Abc47jKL+00123 ABC+47jkL123
```

Sample Output

```
1 -1
2 1
3 0
4 -1
5 0
```

Southeastern European Regional Programming Contest
Bucharest, Romania
October 17, 2009

Problem H
The Lucky Numbers

Input File: H.IN
Output File: standard output
Program Source File: H.C, H.CPP, H.JAVA

John and Brus are freshmen at the primary school. Their first home task is to learn some integer numbers. It is not so hard for the guys and they decide to impress their teacher by learning all lucky numbers between **A** and **B,** inclusive.

As you already know from the previous year contest **4** and **7** are lucky digits, and all the other digits are not lucky. A lucky number is a number that contains only lucky digits in decimal notation.

After learning all lucky numbers in [**A, B**] range John and Brus still have some free time and now they decide to learn additionally each lucky number **N** that is out of [**A, B**] range, but the reversed number of **N** is in this range. Here reversed number of **N** is the number **N** written in decimal notation, but the order of digits is reversed. For example, the reversed number of **447** is **744** and reversed number of **774474444** is **444474477**.

You are given integers **A** and **B** and your task is to find the total number of lucky numbers learned by John and Brus.

Input:

The first line contains single integer **T** – the number of test cases. Each test case consists of a single line containing two integers **A** and **B** separated by a single space.

Output:

For each test case print a single line containing the total number of lucky numbers learned by John and Brus.

Constraints:

$1 \le T \le 74$,

$1 \le A \le B \le$ 100000000000000000000000000000000000000000000000 (10^{47}).

Sample:

Input	Output

2
1 100
44 47
	6
	3

F

[I] Kind of a Blur

Program:	`blur.(c|cpp|java)`
Input:	`blur.in`
Balloon Color:	`Silver`

Description

Image blurring occurs when the object being captured is out of the camera's focus. The top two figures on the right are an example of an image and its blurred version. Restoring the original image given only the blurred version is one of the most interesting topics in image processing. This process is called deblurring, which will be your task for this problem.
In this problem, all images are in grey-scale (no colours). Images are represented as a 2 dimensional matrix of real numbers, where each cell corresponds to the brightness of the corresponding pixel. Although not mathematically accurate, one way to describe a blurred image is through averaging *all the pixels that are within (less than or equal to) a certain Manhattan distance*[†] *from each pixel (including the pixel itself).* Here's an example of how to calculate the blurring of a 3x3 image with a blurring distance of 1:

Focused Image

Blurred Image

6	5	4	3	4	5	6
5	4	3	2	3	4	5
4	3	2	1	2	3	4
3	2	1	0	1	2	3
4	3	2	1	2	3	4
5	4	3	2	3	4	5
6	5	4	3	4	5	6

Manhattan Distance

$$\text{blur}\left(\begin{bmatrix} 2 & 30 & 17 \\ 25 & 7 & 13 \\ 14 & 0 & 35 \end{bmatrix}\right)$$

$$= \begin{bmatrix} \frac{2+30+25}{3} & \frac{2+30+17+7}{4} & \frac{30+17+13}{3} \\ \frac{2+25+7+14}{4} & \frac{30+25+7+13+0}{5} & \frac{17+7+13+35}{4} \\ \frac{25+14+0}{3} & \frac{7+14+0+35}{4} & \frac{13+0+35}{3} \end{bmatrix}$$

$$= \begin{bmatrix} 19 & 14 & 20 \\ 12 & 15 & 18 \\ 13 & 14 & 16 \end{bmatrix}$$

Given the blurred version of an image, we are interested in reconstructing the original version assuming that the image was blurred as explained above.

[†]The Manhattan Distance (sometimes called the Taxicab distance) between two points is the sum of the (absolute) difference of their coordinates. The grid on the lower right illustrates the Manhattan distances from the grayed cell.

Input Format

Input consists of several test cases. Each case is specified on $H+1$ lines. The first line specifies three non negative integers specifying the width W, the height H of the blurred image and the blurring distance D respectively where $(1 \le W, H \le 10)$ and $(D \le \min(W/2, H/2))$. The remaining H lines specify the gray-level of each pixel in the blurred image. Each line specifies W non-negative real numbers given up to the 2^{nd} decimal place. The value of all the given real numbers will be less than 100.

Zero or more lines (made entirely of white spaces) may appear between cases. The last line of the input file consists of three zeros.

Output Format

For each test case, print a $W \times H$ matrix of real numbers specifying the deblurred version of the image. Each element in the matrix should be approximated to 2 decimal places and right justified in a field of width 8. Separate the output of each two consecutive test cases by an empty line. Do not print an empty line after the last test case. It is guaranteed that there is exactly one unique solution for every test case.

Sample Input/Output

blur.in
```
2 2 1
1 1
1 1

3 3 1
19 14 20
12 15 18
13 14 16

4 4 2
14 15 14 15
14 15 14 15
14 15 14 15
14 15 14 15

0 0 0
```

OUTPUT
```
    1.00    1.00
    1.00    1.00

    2.00   30.00   17.00
   25.00    7.00   13.00
   14.00    0.00   35.00

    1.00   27.00    2.00   28.00
   21.00   12.00   17.00    8.00
   21.00   12.00   17.00    8.00
    1.00   27.00    2.00   28.00
```

H · The Two Note Rag

Since most computers are binary machines, both powers of two and problems that involve only two values are important to computer scientists. The following problem has to do with powers of two and the digits **1** and **2**.

Some powers of two as decimal values, such as 2^9 = **512** and 2^{89} = **618,970,019,642,690,137,449,562,112** end in a string of digits consisting only of **1**'s and **2**'s (**12** for 2^9 and **2112** for 2^{89}). In fact, it can be proved that:

*For every integer **R**, there exists a power of **2** such that 2^K uses **only** the digits **1** and **2** in its last **R** digits.*

This is shown a bit more clearly in the following table:

R	Smallest K	2^K
1	1	2
2	9	512
3	89	…112
4	89	…2112
5	589	…22112
6	3089	…122112
…	…	…

Your job is to write a program that will determine, for given **R**, the *smallest* **K** such that 2^K ends in a string of **R** digits containing only **1**'s and **2**'s.

Input

The first line of the input contains a single decimal integer, ***N***, $1 \le N \le 50$, the number of problem data sets to follow. Each data set consists of a single integer ***R***, $1 \le R \le 20$, for which we want a power of **2** ending in a string of ***R*** **1**'s and **2**'s.

Output

For each data set, you should generate one line of output with the following values: The data set number as a decimal integer (start counting at one), a space, the input value ***R***, another space, and the smallest value ***K*** for which 2^K ends in a string of ***R*** **1**'s and **2**'s.

Sample Input	Sample Output
6	1 1 1
1	2 2 9
2	3 4 89
4	4 5 589
5	5 7 3089
7	6 15 11687815589
15	

Southeastern European Regional Programming Contest
Bucharest, Romania
October 17, 2009

Problem I
The Robbery

Input File: I.IN
Output File: standard output
Program Source File: I.C, I.CPP, I.JAVA

In the downtown of Bucharest there is a very big bank with a very big vault. Inside the vault there are **N** very big boxes numbered from **1** to **N**. Inside the box with number **k** there are **k** very big diamonds, each of weight $\mathbf{W_k}$ and cost $\mathbf{C_k}$.

John and Brus are inside the vault at the moment. They would like to steal everything, but unfortunately they are able to carry diamonds with the total weight not exceeding **M**.

Your task is to help John and Brus to choose diamonds with the total weight less than or equal to **M** and the maximal possible total cost.

Input: standard input

The first line contains single integer **T** – the number of test cases. Each test case starts with a line containing two integers **N** and **M** separated by a single space. The next line contains **N** integers $\mathbf{W_k}$ separated by single spaces. The following line contains **N** integers $\mathbf{C_k}$ separated by single spaces.

Output: standard output

For each test case print a single line containing the maximal possible total cost of diamonds.

Constraints:

$1 \le T \le 74$,
$1 \le N \le 15$,
$1 \le M \le 1000000000$ (10^9),
$1 \le W_k, C_k \le 1000000000$ (10^9).

Sample:

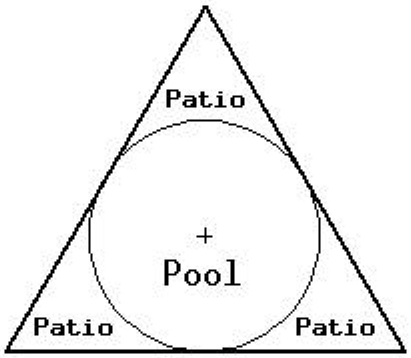
Input	Output

2
2 4
3 2
5 3
3 100
4 7 1
5 9 2
	6
	29

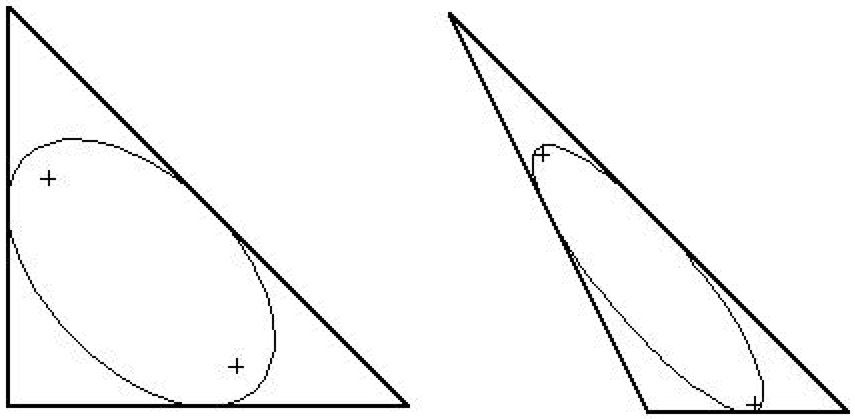
I

I · Joe's Triangular Gardens

Joe's landscaping company specializes in gardens for computer geeks who have just had their company go public. One of his signature features is a round pool surrounded by a tiled patio in the form of an equilateral triangle where the edge of the pool is tangent to each side of the triangle at its midpoint.



Unfortunately, some of Joe's customers are not satisfied with an equilateral triangle, usually in the center of the garden. Some want it in a corner or next to a slope or some other layout. Joe would like the option of offering arbitrary triangular patios with an elliptical pool which is tangent to each side at the center of the side. For example:



Joe knows how to draw an ellipse by putting two stakes in the ground (at the foci of the ellipse), tying a rope between them and dragging a marker stick inside the rope. What Joe would like is for the customer to determine where the corners of the triangle will be and then measure the location of the triangle vertices and compute where to put the stakes and how long to make the rope.

Write a program, which takes as input the three vertices (**x1**, **y1**), (**x2**, **y2**) and (**x3**, **y3**) of a triangle and computes an ellipse inscribed in the triangle, which is tangent to each side of the triangle at its midpoint. The output is the coordinates of the two foci of the ellipse and the length of the rope (which is the sum of the distances from the foci to any point on the ellipse.

Input

The first line of input contains a single integer ***N***, (1 ≤ ***N*** ≤ 1000) which is the number of data sets that follow. Each data set consists of a single line of input containing 6 space separated floating point numbers **x1 y1 x2 y2 x3 y3** giving the coordinates of the vertices of a triangle.

Output

For each data set, you should generate one line of output with the following values: The data set number as a decimal integer (start counting at one), a space and five floating point values accurate to two decimal places each separated by a single space. The values are **fx1 fy1 fx2 fy2 rl** where (**fx1**, **fy1**) is one focus of the ellipse, (**fx2**, **fy2**) is the other focus of the ellipse and **rl** is the sum of the distances from the foci to any point on the ellipse (e.g. the length of the rope). The foci should be listed in *increasing lexicographical order* (i.e. **fx1** <= **fx2** and if **fx1** = **fx2**, **fy1** <= **fy2**). Note that in the case the ellipse is a circle, the two foci are the same (e.g. the center of the circle).

Sample Input	Sample Output

3
100 100 200 273.2051 300 100
100 100 100 300 300 100
100 200 100 300 300 100
	1 200.00 157.71 200.00 157.76 115.47
	2 119.53 213.81 213.81 119.53 163.30
	3 103.94 253.14 229.40 146.86 170.51

Southeastern European Regional Programming Contest
Bucharest, Romania
October 17, 2009

Problem J
The Computer Game

Input File: J.IN
Output File: standard output
Program Source File: J.C, J.CPP, J.JAVA

John and Brus are playing a military strategic game on a PC. The game is played on the flat world map. At the beginning of the game Brus places his army. Then John has to choose strategic points for his army according to the following rules:

- each strategic point must be a lattice point (**x, y**) (a lattice point is a point with integer coordinates) such that **|x|** + **|y|** < **N;**
- John can choose any positive number of strategic points;
- all the strategic points must be distinct;
- each of the strategic points must be free (i.e. not occupied by Brus's army);
- each pair of different strategic points must be connected (possibly via some other strategic points).

Here two different lattice points ($\mathbf{x_1}$, $\mathbf{y_1}$) and ($\mathbf{x_2}$, $\mathbf{y_2}$) are connected if $|\mathbf{x_1} - \mathbf{x_2}| + |\mathbf{y_1} - \mathbf{y_2}| = \mathbf{1}$. If **A**, **B** and **C** are strategic points, **A** and **B** are connected, **B** and **C** are connected, then **A** and **C** are also connected.

Your task is to find the number of ways for John to choose strategic points for his army.

Input

The first line contains single integer **T** – the number of test cases. Each test case starts with a line containing two integers **N** and **M** separated by a single space. **N** is the number mentioned in the first rule. **M** is the number of lattice points on the world map already occupied by Brus's army. Each of the following **M** lines contains two integers $\mathbf{X_k}$ and $\mathbf{Y_k}$ separated by a single space. Each lattice point ($\mathbf{X_k}$, $\mathbf{Y_k}$) is occupied by Brus's army.

Output

For each test case print a single line containing the number of ways for John to choose strategic points for his army.

Constraints:
$1 \le T \le 74$,
$1 \le N \le 7$,
$1 \le M \le 225$,
$-7 \le X_k, Y_k \le 7$,
all ($\mathbf{X_k}$, $\mathbf{Y_k}$) will be distinct.

Sample:

Input	Output

2
2 1
7 7
2 3
0 0
4 -7
7 -4
	20
	4

K