10219 - Find the ways !

Time limit: 3.000 seconds

An American, a Frenchman and an Englishwoman had been to Dhaka, the capital of Bangladesh. They went sight-seeing in a taxi. The three tourists were talking about the sites in the city. The American was very proud of tall buildings in New York. He boasted to his friends, "Do you know that the Empire State Building was built in three months?"

"Really?" replied the Frenchman. "The Eiffel Tower in Paris was built in only one month! (However, The truth is, the construction of the Tower began in January 1887. Forty Engineers and designers under Eiffel's direction worked for two years. The tower was completed in March 1889.)

"How interesting!" said the Englishwoman. "Buckingham Palace in London was built in only two weeks!!"

At that moment the taxi passed a big slum (However, in Bangladesh we call it "Bostii"). "What was that? When it was built ?" The Englishwomen asked the driver who was a Bangladeshi.

"I don't know!" , answered the driver. "It wasn't there yesterday!"

However in Bangladesh, illegal establishment of slums is a big time problem. Government is trying to destroy these slums and remove the peoples living there to a far place, formally in a planned village outside the city. But they can't find any ways, how to destroy all these slums!

Now, can you imagine yourself as a slum destroyer? In how many ways you can destroy ***k*** slums out of ***n*** slums ! Suppose there are 10 slums and you are given the permission of destroying 5 slums, surly you can do it in 252 ways, which is only a 3 digit number, Your task is to find out the digits in ways you can destroy the slums !

The Input

The input file will contain one or more test cases.

Each test case consists of one line containing two integers n (n>=1) and k (1<=<k=<n).

The Output

For each test case, print one line containing the required number. This number will always fit into an integer, i.e. it will be less than 2^31-1.

Sample Input

```
20 5
100 10
200 15
```

Sample Output

```
5
14
23
```

167 - The Sultan's Successors

Time limit: 3.000 seconds

The Sultan of Nubia has no children, so she has decided that the country will be split into up to *k* separate parts on her death and each part will be inherited by whoever performs best at some test. It is possible for any individual to inherit more than one or indeed all of the portions. To ensure that only highly intelligent people eventually become her successors, the Sultan has devised an ingenious test. In a large hall filled with the splash of fountains and the delicate scent of incense have been placed *k* chessboards. Each chessboard has numbers in the range 1 to 99 written on each square and is supplied with 8 jewelled chess queens. The task facing each potential successor is to place the 8 queens on the chess board in such a way that no queen threatens another one, and so that the numbers on the squares thus selected sum to a number at least as high as one already chosen by the Sultan. (For those unfamiliar with the rules of chess, this implies that each row and column of the board contains exactly one queen, and each diagonal contains no more than one.)

Write a program that will read in the number and details of the chessboards and determine the highest scores possible for each board under these conditions. (You know that the Sultan is both a good chess player and a good mathematician and you suspect that her score is the best attainable.)

Input

Input will consist of *k* (the number of boards), on a line by itself, followed by *k* sets of 64 numbers, each set consisting of eight lines of eight numbers. Each number will be a positive integer less than 100. There will never be more than 20 boards.

Output

Output will consist of *k* numbers consisting of your *k* scores, each score on a line by itself and right justified in a field 5 characters wide.

Sample input

```
1
 1  2  3  4  5  6  7  8
 9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
48 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
```

Sample output

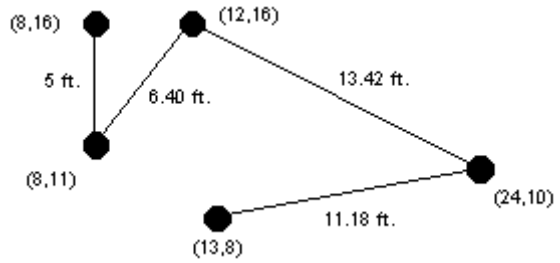
```
  260
```

216 - Getting in Line Time limit: 3.000 seconds

Computer networking requires that the computers in the network be linked.

This problem considers a ``linear" network in which the computers are chained together so that each is connected to exactly two others except for the two computers on the ends of the chain which are connected to only one other computer. A picture is shown below. Here the computers are the black dots and their locations in the network are identified by planar coordinates (relative to a coordinate system not shown in the picture).

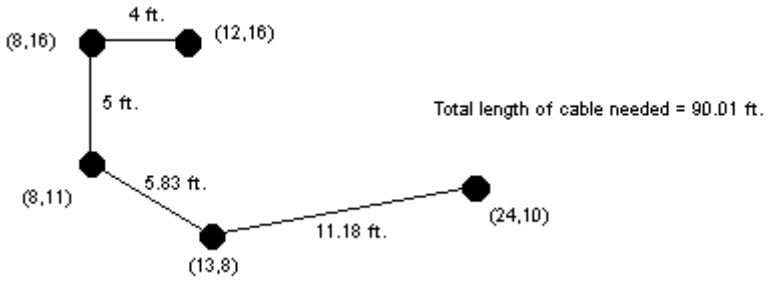
Distances between linked computers in the network are shown in feet.



For various reasons it is desirable to minimize the length of cable used.

Your problem is to determine how the computers should be connected into such a chain to minimize the total amount of cable needed. In the installation being constructed, the cabling will run beneath the floor, so the amount of cable used to join 2 adjacent computers on the network will be equal to the distance between the computers plus 16 additional feet of cable to connect from the floor to the computers and provide some slack for ease of installation.

The picture below shows the optimal way of connecting the computers shown above, and the total length of cable required for this configuration is (4+16)+ (5+16) + (5.83+16) + (11.18+16) = 90.01 feet.



Input

The input file will consist of a series of data sets. Each data set will begin with a line consisting of a single number indicating the number of computers in a network. Each network has at least 2 and at most 8 computers. A value of 0 for the number of computers indicates the end of input.

After the initial line in a data set specifying the number of computers in a network, each additional line in the data set will give the coordinates of a computer in the network. These

coordinates will be integers in the range 0 to 150. No two computers are at identical locations and each computer will be listed once.

Output

The output for each network should include a line which tells the number of the network (as determined by its position in the input data), and one line for each length of cable to be cut to connect each adjacent pair of computers in the network. The final line should be a sentence indicating the total amount of cable used.

In listing the lengths of cable to be cut, traverse the network from one end to the other. (It makes no difference at which end you start.) Use a format similar to the one shown in the sample output, with a line of asterisks separating output for different networks and with distances in feet printed to 2 decimal places.

Sample Input

```
6
5 19
55 28
38 101
28 62
111 84
43 116
5
11 27
84 99
142 81
88 30
95 38
3
132 73
49 86
72 111
0
```

Sample Output

```
**********************************************************
Network #1
Cable requirement to connect (5,19) to (55,28) is 66.80 feet.
Cable requirement to connect (55,28) to (28,62) is 59.42 feet.
Cable requirement to connect (28,62) to (38,101) is 56.26 feet.
Cable requirement to connect (38,101) to (43,116) is 31.81 feet.
Cable requirement to connect (43,116) to (111,84) is 91.15 feet.
Number of feet of cable required is 305.45.
**********************************************************
Network #2
Cable requirement to connect (11,27) to (88,30) is 93.06 feet.
Cable requirement to connect (88,30) to (95,38) is 26.63 feet.
Cable requirement to connect (95,38) to (84,99) is 77.98 feet.
Cable requirement to connect (84,99) to (142,81) is 76.73 feet.
Number of feet of cable required is 274.40.
**********************************************************
Network #3
Cable requirement to connect (132,73) to (72,111) is 87.02 feet.
Cable requirement to connect (72,111) to (49,86) is 49.97 feet.
Number of feet of cable required is 136.99.
```

10306 - e-Coins

Time limit: 3.000 seconds

At the Department for Bills and Coins, an extension of today's monetary system has newly been proposed, in order to make it fit the new economy better. A number of new so called e-coins will be produced, which, in addition to having a value in the normal sense of today, also have an InfoTechnological value. The goal of this reform is, of course, to make justice to the economy of numerous dotcom companies which, despite the fact that they are low on money surely have a lot of **IT** inside. All money of the old kind will keep its conventional value and get zero InfoTechnological value.

To successfully make value comparisons in the new system, something called the e-modulus is introduced. This is calculated as **SQRT(X*X+Y*Y)**, where **X** and **Y** hold the sums of the conventional and InfoTechnological values respectively. For instance, money with a conventional value of **$3** altogether and an InfoTechnological value of **$4** will get an e-modulus of **$5**. Bear in mind that you have to calculate the sums of the conventional and InfoTechnological values separately before you calculate the e-modulus of the money.

To simplify the move to e-currency, you are assigned to write a program that, given the e-modulus that shall be reached and a list of the different types of e-coins that are available, calculates the smallest amount of e-coins that are needed to exactly match the e-modulus. There is no limit on how many e-coins of each type that may be used to match the given e-modulus.

Input

A line with the number of problems **n (0<n<=100)**, followed by **n** times:

- A line with the integers **m (0<m<=40)** and **S (0<S<=300)**, where **m** indicates the number of different e-coin types that exist in the problem, and **S** states the value of the e-modulus that shall be matched exactly.
- **m** lines, each consisting of one pair of non-negative integers describing the value of an e-coin. The first number in the pair states the conventional value, and the second number holds the InfoTechnological value of the coin.

When more than one number is present on a line, they will be separated by a space. Between each problem, there will be one blank line.

Output

The output consists of **n** lines. Each line contains either a single integer holding the number of coins necessary to reach the specified e-modulus **S** or, if **S** cannot be reached, the string **"not possible"**.

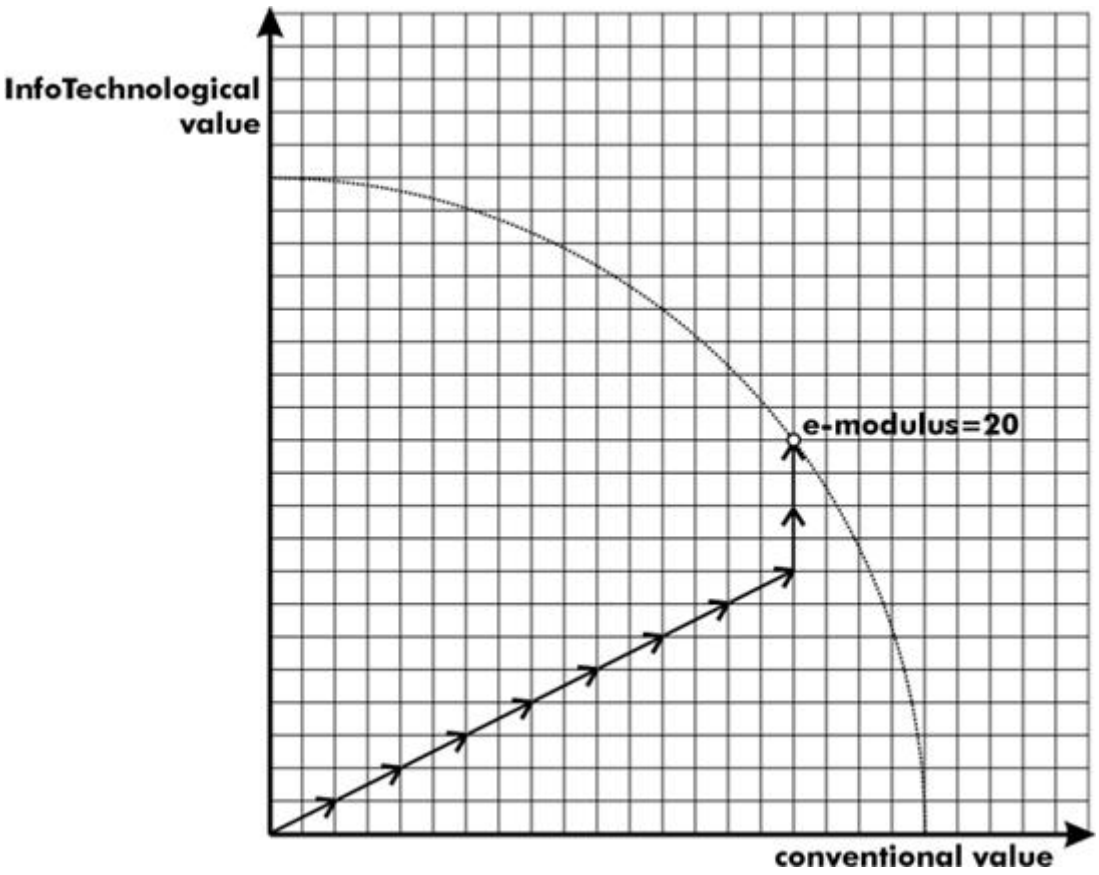
Sample Input:

```
3
2 5
0 2
2 0

3 20
0 2
2 0
2 1

3 5
3 0
0 4
5 5
```

Sample Output:

```
not possible
10
2
```

(Joint Effort Contest, Problem Source: Swedish National Programming Contest, arranged by department of Computer Science at Lund Institute of Technology.)



191 - Intersection

Time limit: 3.000 seconds

You are to write a program that has to decide whether a given line segment intersects a given rectangle.

An example:

```
line:           start point:          (4,9)

                end point:            (11,2)

rectangle:              left-top:             (1,5)

                right-bottom:         (7,1)
```

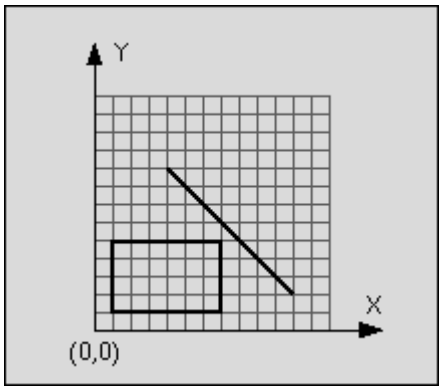


Figure: Line segment does not intersect rectangle

The line is said to intersect the rectangle if the line and the rectangle have at least one point in common. The rectangle consists of four straight lines and the area in between. Although all input values are integer numbers, valid intersection points do not have to lay on the integer grid.

Input

The input consists of *n* test cases. The first line of the input file contains the number *n*. Each following line contains one test case of the format:

xstart ystart xend yend xleft ytop xright ybottom

where (*xstart*, *ystart*) is the start and (*xend*, *yend*) the end point of the line and (*xleft*, *ytop*) the top left and (*xright*, *ybottom*) the bottom right corner of the rectangle. The eight numbers are

separated by a blank. The terms *top left* and *bottom right* do not imply any ordering of coordinates.

Output

For each test case in the input file, the output file should contain a line consisting either of the letter "`T`" if the line segment intersects the rectangle or the letter "`F`" if the line segment does not intersect the rectangle.

Sample Input

```
1
4 9 11 2 1 5 7 1
```

Sample Output

```
F
```

677 - All Walks of length "n" from the first node

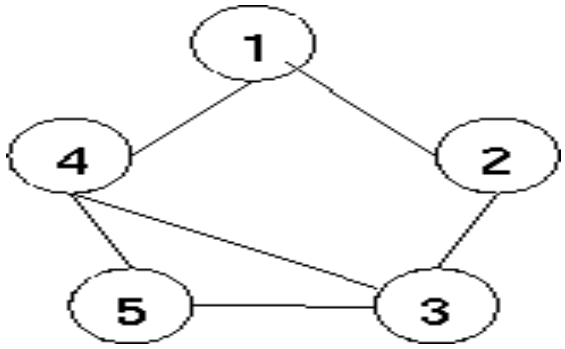
Time limit: 3.000 seconds

A computer network can be represented as a graph. Let G = (V, E) be an undirected graph, V = $(v_1, v_2, v_3, \ldots, v_m)$ represents all nodes, where *m* is the number of nodes, and E represents all edges. The first node is v_1 and the last node is v_m . The number of edges is *k*. Define the adjacency matrix $A = (a_{ij})_{m \times m}$ where

$$a_{ij} = 1 \quad \text{if } \{v_i, v_j\} \in \mathbf{E}, \text{ otherwise} \quad a_{ij} = 0$$

An example of the adjacency matrix and its corresponding graph are as follows:

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$



Calculate

$$A^n = \underbrace{A \cdot A \cdots A}_{n}$$

and use the Boolean operations where 0+0 = 0, 0+1=1+0=1, 1+1=1, and $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0, 1 \cdot 1 = 1$. The entry in row *i* and column *j* of A_n is 1 if and only if at least there exists a walk of length *n* between the *i*-th and *j*-th nodes of V. In other words, the distinct walks of length *n* between the *i*-th and *j*-th nodes of V may be more than one. Note that the node in the paths can be repetitive.

The following example shows the walks of length 2.

$$A^2 = A \cdot A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Write programs to do above calculation and print out all distinct walks of length *n*. (In this problem we let the maximum walks of length *n* be 5 and the maximum number of nodes be 10.)

Input

The input file contains a number of test examples, the test examples are separated by `-9999`. Each test example consists of the number of nodes and the length of walks in the first row, and then the adjacency matrix.

Output

The output file must contain all distinct walks of the length *n*, and with all its nodes different, from the first node, listed in lexicographical order. In case there are not walks of length *n*, just print `` `no walk of length n' ``

Separate the output of the different cases by a blank line.

Sample Input

```
5 2
0 1 0 1 0
1 0 1 0 0
0 1 0 1 1
1 0 1 0 1
0 0 1 1 0
-9999
5 3
0 1 0 1 0
1 0 1 0 0
0 1 0 1 1
1 0 1 0 1
0 0 1 1 0
```

Sample Output

```
(1,2,3)
(1,4,3)
(1,4,5)

(1,2,3,4)
(1,2,3,5)
(1,4,3,2)
(1,4,3,5)
(1,4,5,3)
```