Problem A:

Extreme Fibonacci

Source file: fib.{c|cpp|java}
Input file: fib.in

The Fibonnacci sequence is a well known sequence invented in the 15th century to describe the population growth of rabbits. The sequence is given by the following recurrence formula:

$F_0=1$
$F_1=1$
$F_n = F_{n-1} + F_{n-2}$ for $n\geq 2$.

Thus, the first Fibonacci terms are 1, 1, 2, 3, 5, 8, 13, ...

Prof. Goingnuts is trying to study further the sequence by calculating the last digits of any number in the sequence.

Your assigned task is that given two integers n and k, you find the k least significant digits of the F_n term.

Input

The input file begins with a single positive integer M which is the number of cases to examine. M lines follow, each consisting of two number n and k separated by a single space, where $1\leq n\leq 10000$ and $1\leq k\leq 9$.

Output

For each n and k pair, the corresponding k digits should be output on a line of their own. Your output should include any leading zeros that are necessary to complete the k digits.

Sample Input (*fib.in*)

```
4
1 1
5 3
100 4
1000 5
```

Sample Output *(standard output)*

```
1
008
4101
03501
```

Problem B:

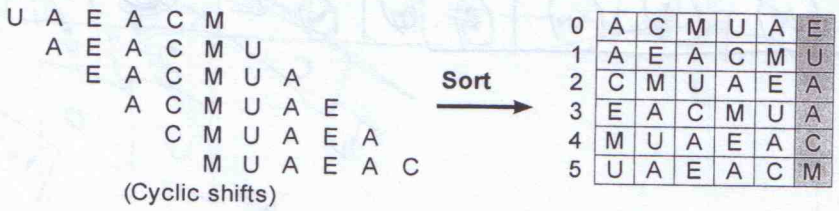
Burrows-Wheeler Transform

Source file: bw.{c|cpp|java}
Input file: bw.in

The Burrows-Wheeler transform is a fancy text manipulation technique that is used as a pre-processing step in text compression. The tranform (which by the way is fully reversible) works by rearranging the characters in a string of length N as follows:

1. All the N possible cyclic shifts of the string are produced. Recall that a circular shift is a permutation of the entries in a tuple where the first element become the last element and all the other are shifted
2. The strings are sorted.
3. The last character from each of the sorted strings is used to form the output string.
4. The position of the original string in the sorted list is output.

For example, for string "UAEACM", the transformation would produce the following sorted list:



and the output would be: 5, "EUAACM". Please note that all the letters of the original string exist in the transformation, although with a different order.

Your assigned task is to find the Burrows-Wheeler transform of a given string.

Input

The input file begin with a single positive integer M which is the number of cases to examine. M lines follow, each containing a string of alphanumerical symbols (including white-space and punctuation marks). Each string can be up to 10000 symbols longs.

Output

For each input string you should output two lines. The first one should contain the location of the input string in the sorted sequence and the second one the transformed string. The output for two successive input cases should be separated by a single line.

Sample Input *(bw.in)*

```
2
UAEACM
Burrows-Wheeler Transform
```

Sample Output (standard output)

```
5
EUAACM

2
rsm -rhelsWerafreToruwnBo
```

Problem C:

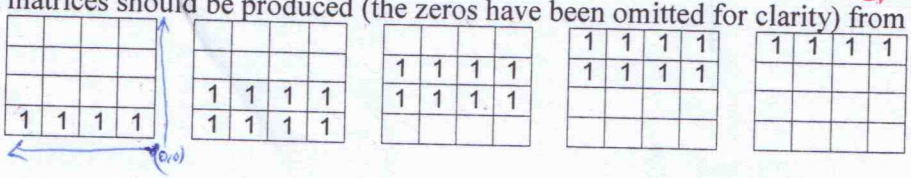
Disco Floor

Source file: disco.{c|cpp|java}
Input file: disco.in

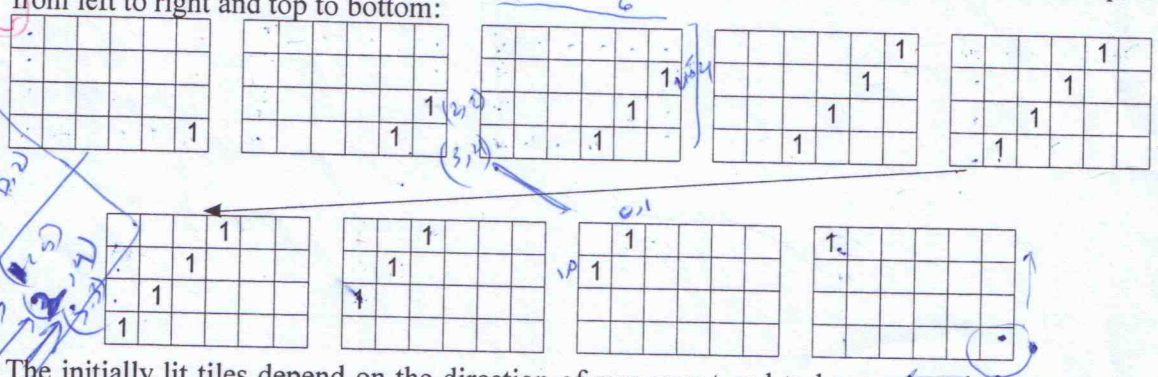
The administration in your university has taken one more step towards enhancing your learning experience by building a new disco where students can hang out and take a rest from classes and assignments. The trouble is that the newly installed floor is not working properly.

The floor consists of a rectangular grid of NxM tiles that can be individually lit. This can be done by writing a binary matrix of NxM elements to an appropriate hardware port. To make the DJ's life easier, you are to write a program that accepts commands that light the tiles in a predetermined fashion, by producing the sequence of necessary matrices.

The commands consist of a direction of movement (one of N (☐), S(☐), W(☐), E(☐), NE(☐), NW(☐), SE(☐) or SW(☐)) and the number of tile diagonals/rows/columns that should be simultaneously lit. If the tiles were considered to form a line, the second part of the command constitutes the line thickness. For example, for a 4x4 floor and the command "N 2" the following sequence of matrices should be produced (the zeros have been omitted for clarity) from left to right:



For a 4x6 floor and the command "NW 1" the following sequence of matrices should be produced from left to right and top to bottom:



The initially lit tiles depend on the direction of movement and to be precise, they are on the exact opposite location of the direction of movement. To add to the above examples, a direction of S would mean that the north row of tiles will be initially lit. If the direction is SW, the top-right corner tile will be initially lit, etc.

Input

The input file begin with a single positive integer M which is the number of cases to examine. Each of the input cases starts with a line containing 3 integers, x, y and n separated by a single space. The first two represent the number of columns and rows respectively, while n is the number of floor lighting instructions. n lines follow, each containing the direction of movement (in uppercase letters) and line thickness.

Output

The output for each of the input cases and each command string, consists of a table of '.' and 'O' characters, the latter representing a lit tile. Successive tables are separated by a single empty line.

Sample Input *(disco.in)*

```
2
4 2 1
E 4
5 1 2
S 2
SE 2
```

Sample Output *(standard output)*

```
O...
O...

OO..
OO..

OOO.
OOO.

OOOO
OOOO

.OOO
.OOO

..OO
..OO

...O
...O

OOOOO

OOOOO

O....

OO...

.OO..

..OO.

...OO

....O
```

Problem D:

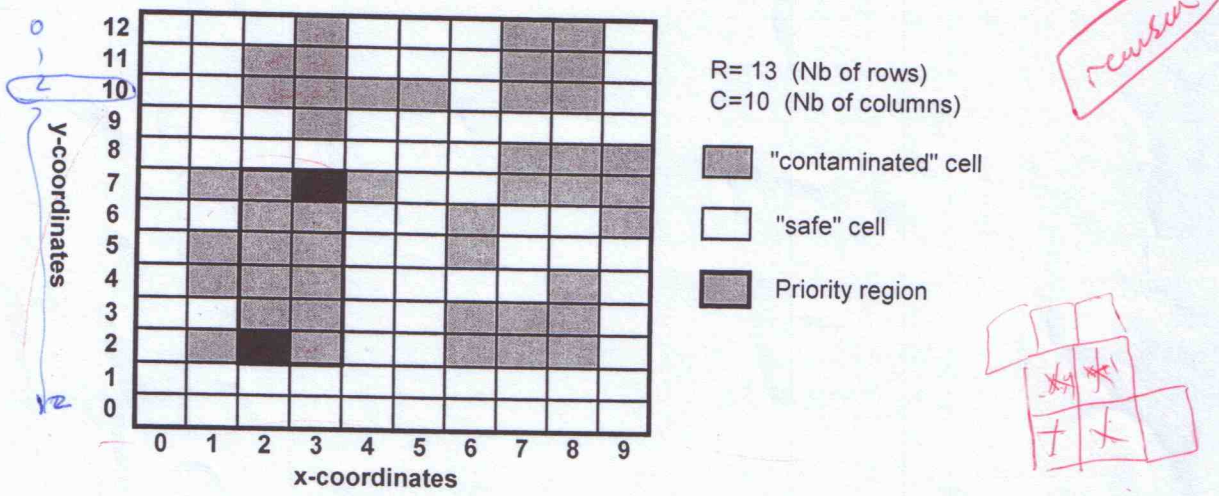
LEAKY Oil Spill Rescue Operation

Source file: leaky.{c|cpp|java}
Input file: leaky.in

Every year, hundreds of millions of gallons of oil quietly end up in the sea. Oil spills can have devastating effect on marine animals and corals beaches. As a result, different technologies and methods have been developed to assist in monitoring and containing oil spills.

Recently, Geographic Information Systems (GIS), along with remote sensing devices, such as space-borne synthetic aperture radars (SARs), have been used to assist rescue teams in the containment of marine oil spills.

The National coast guards have recently been informed that the oil tanker *LEAKY* ran aground off the Arabian Gulf, spilling few million of gallons of oil. SAR images of the see surface in the neighborhood of the spill area have been collected, filtered, and then processed to produce a flat binary file from raster-based spatial data. As shown in the sample case below, a shaded cell represents a "contaminated" see area and is encoded as a binary "1" in the flat binary file. Similarly, a clear cell represents a spill-free (safe) area, and is encoded as a binary "0".



Raster-based spatial data for the oil spill

The first task of the rescue team was to identify the ***largest*** rectangular spill area so that it can be surrounded as soon as possible by a rectangular boom (bracket) to collect the leaked oil. The constraint on the rectangular geometry of the boom is dictated by some technical, as well as operational considerations. The identification of this (all ones) target area (contoured in bold in the figure above) will help the team prioritize the cleaning operation.

The rescue team is soliciting your assistance to help identify the priority region as defined above. This priority region will be identified by the x and y coordinates of the upper-right and lower-left corners. These two corner cells are dark-shaded in the above example.

Input

The input to this problem starts with a number M describing how many cases are to be examined. Each case starts with a line containing the corresponding number of rows (R) and the number of columns (C), separated by a single space (You can assume that

these 2 numbers are ≤ 100). This is followed by the corresponding (RxC) Raster grid. The grid entries are valid binary digits, separated by a single space.

Output

For each input case, you have to output the coordinates of the priority region, in the sequence/format given below (see sample output case for details).

- The **x** coordinate of the lower-left corner
- The **y** coordinate of the lower-left corner
- The **x** coordinate of the upper-right corner
- The **y** coordinate of the upper-right corner, **as well as**
- The number of contaminated cells inside the identified priority area.

Successive outputs for different cases will be separated by a single empty line.
Note: if the grid has no contaminated cell, the program should set the x and y coordinates of the lower-left corner and the upper-right corner to (0,0) and (-1,-1), respectively. Further, if there is more than one priority region, pick-up any one.

Sample Input *(leaky.in)* (This first input corresponds to the case displayed in the figure above)

```
2
13 10
0 0 0 1 0 0 0 1 1 0
0 0 1 1 0 0 0 1 1 0
0 0 1 1 1 1 0 1 1 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1
0 1 1 1 1 0 0 1 1 1
0 0 1 1 0 0 0 0 1 1
0 1 1 1 0 1 0 0 0 0
0 1 1 1 0 0 0 0 1 0
0 0 1 1 0 0 1 1 1 0
0 1 1 1 0 0 1 1 1 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
 8 9
 0 0 1 1 1 1 1 1 1
 1 1 1 0 0 0 0 0 0
 1 0 1 1 0 0 0 1 1
 1 1 1 1 1 0 0 1 1
 0 0 0 0 0 0 0 1 1
 1 1 0 0 1 1 0 0 0
 1 1 0 0 0 0 0 0 1
 1 0 0 1 0 0 1 1 1
```

Sample Output *(standard output)*

```
Lower_left_X= 2
Lower_left_Y= 2
Upper_right_X= 3
Upper_right_Y= 7
The number of contaminated cells is 12

Lower_left_X= 2
Lower_left_Y= 7
Upper_right_X= 8
Upper_right_Y= 7
The number of contaminated cells is 7
```

Problem E:

Neighbors

Source file: neighbors.{c|cpp|java}
Input file: neighbors.in

We need to find out the neighbors of a cell in a given two-dimensional array of dimension N*M (N rows and M columns). The neighbors could either be north, south, west or east. If a cell does not have neighbors in a particular direction (e.g. North) your program must print "null" as the neighbor.

Input

The input to this program is an arbitrary two dimensional array matrix of size (N*M) elements to be examined. The first line in the input file gives the dimensions of the array: number of lines (*N*) and the number of columns (*M*), separated by a single space (e.g. 5 5) followed by k consecutive lines ($k \leq N * M$). Each line contains the position (coordinates) of the elements for which you need to find out the neighbors.

Output

For each element stated in the last lines for which their positions (indexes) are given, we display their neighbours. Note that the index for each line start from 0 and the index for each column start with 0 as well. According to the example stated in the sample input the neighbors of the element of position (1,1) that has a value of 6 are as follow:

- North neighbors: 1
- South neighbors: 2, 7, 3
- West neighbors: 5
- East neighbors: 7, 8, 0

Note that for multiple input cases the output (the list of neighbors) has to be separated by a blank line.

Sample Input *(neighbors.in)*

```
5 5
0 1 2 3 4
5 6 7 8 0
1 2 3 4 5
6 7 8 0 1
2 3 4 5 6
1 1
3 4
```

Sample Output *(standard output)*

```
The west neighbors of element 6 at position [1][1] are: 5 end of list
The east neighbors of element 6 at position [1][1] are: 7 8 0 end of list
The north neighbors of element 6 at position [1][1] are: 1 end of list
The south neighbors of element 6 at position [1][1] are: 2 7 3 end of list
```

Problem F:

Compression

Source file: compression {.c| .cpp| .java}
Input file: compression.in

Many data files (such as pictures) are characterized by large sequences of repeated characters. These files can be compressed using a technique called run-length encoding (RLE). For this problem, you will need to implement a variant of RLE.

Suppose we have a sequence of 20 '.' characters, the basic idea of the original RLE is to replace, the 20 consecutive occurrences of the character '.' with the number 20 and a single '.'. We can also replace sequences of more than one character. For example, the sequence abcabcabcabcabc can be converted into 0503abc where the first two characters, 05, are a pair of digits specifying how many times the pattern is repeated and the following two characters, 03, are the length of the repeated pattern, followed by the sequence of characters.

Each compression has again represented by number of characters saved. In the example above, 12 characters were saved (the digits are not to be counted when considering the length of the compressed sequence). Each string to be compressed will probably consist of more than one repeated pattern. For example, the string abcdabcdabcdefefefefgggggg should be compressed into 0304abcd0402ef0601g.

When compressing a string, we start by compressing the repeated sequence which saves the most characters first, and then compress what's left over on the left, and on the right. For example, the string aabbaabbbbaabb should be compressed as 0201a0206bbaabb and not as 0204aabb0106bbaabb or 0204aabb0202bb0202aa0202bb. If more than one such sequence exists, pick the longer one. If more than one have the same length, pick the one further to the right.

If a string has no repeating sequences, output it with no digits in front.

You will receive a number of lines of text on the input file. These lines will not contain digits. You are to compress each one as described above, and write your answer to standard output. Use one line of output for each line of input. The last line in the input file is blank (i.e. stop reading once a blank line is detected).

Input

Each line of the input contains a set of characters, each containing a string of alphabetical characters (including blanks). Each string can be up to 9999 characters and does not contain numerical digits.

Output

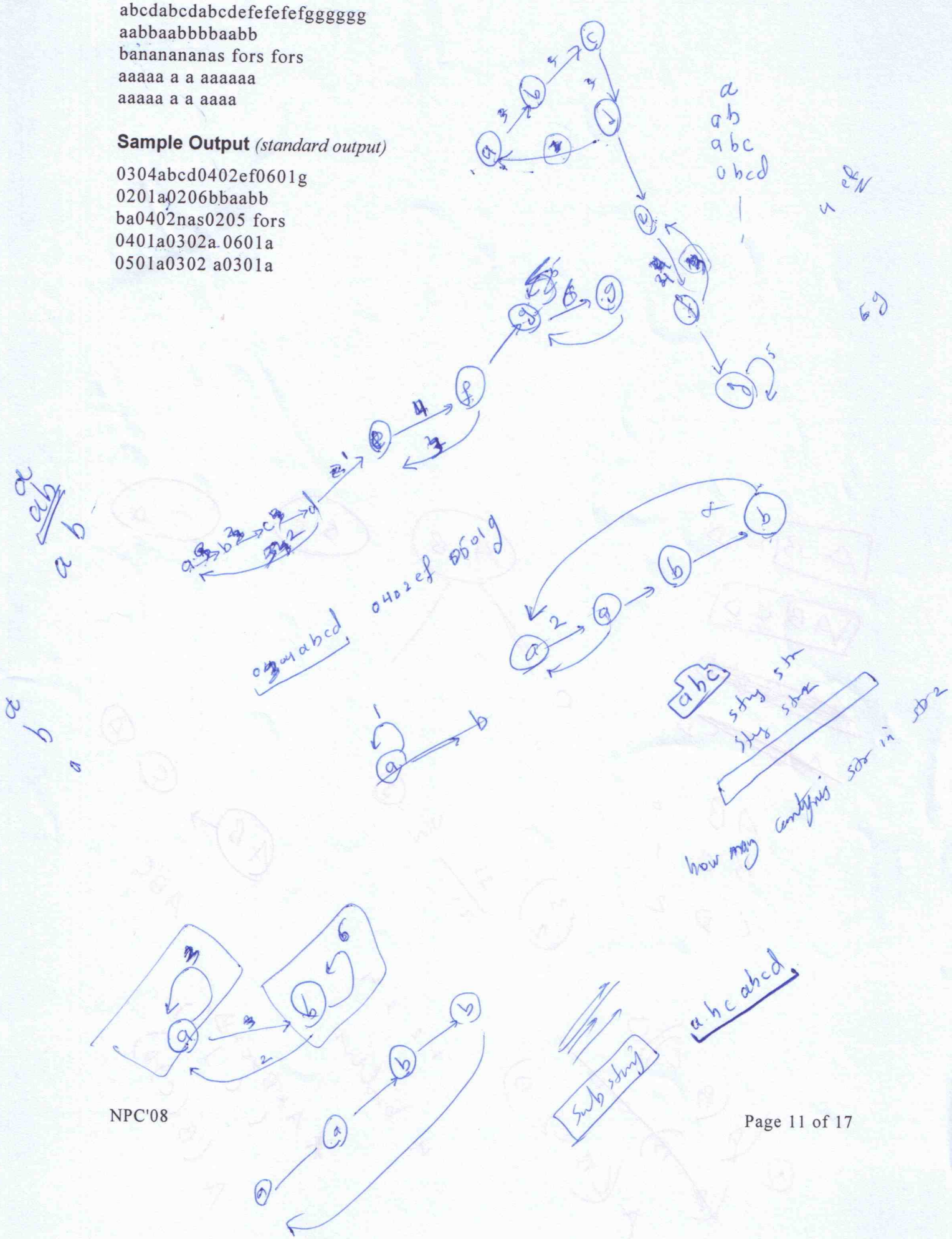
For each input string in each line you should display one output. Each output line contains the compressed string using the RLE algorithm explained above. If an input string has no repeating sequences, the displayed output should be the string itself.

Sample Input *(compression.in)*

abcdabcdabcdefefefefgggggg
aabbaabbbbaabb
bananananas fors fors
aaaaa a a aaaaaa
aaaaa a a aaaa

Sample Output *(standard output)*

0304abcd0402ef0601g
0201a0206bbaabb
ba0402nas0205 fors
0401a0302a 0601a
0501a0302 a0301a



Problem G:

Chained Matrix Multiplication

Source file: matrix {.c| .cpp| .java}
Input file: matrix.in

Suppose that we multiply two matrices (A 2x3 matrix and a 3x4 matrix) as follows:

$$\begin{matrix}1 & 2 & 3\\ 4 & 5 & 6\end{matrix} \quad X \quad \begin{matrix}7 & 8 & 9 & 1\\ 2 & 3 & 4 & 5\\ 6 & 7 & 8 & 9\end{matrix} \quad = \quad \begin{matrix}29 & 35 & 41 & 38\\ 74 & 89 & 104 & 83\end{matrix}$$

The resulting matrix is a 2X4 matrix. In the traditional matrix multiplication, every item (cell) in the resulting matrix requires 3 multiplications. For example, the first cell in the first column is obtained by 1*7 +2*2 + 3*6 = 29. There are 2*4=8 cells or entries in the resulting matrix requiring in total 2*4*3=24 multiplications. In general, if we multiply an iXj matrix by a jXk matrix using the standard method, it is necessary to do i*j*k elementary multiplications.

Matrix multiplication is an associative operation, meaning that the order of multiplications does not matter as long as the sequence of the operands is not changed. That is, rearranging the parentheses in such an expression will not change it value (i.e. the final resulting matrix is always the same) but the number of elementary multiplication operations can be very different. We want to multiply a set of matrices using the best possible order. Consider for example the following set (ABCD) of matrices presented in Table 1. Table2 shows the number of multiplications that are needed using various orders; clearly the third order is the optimal one.

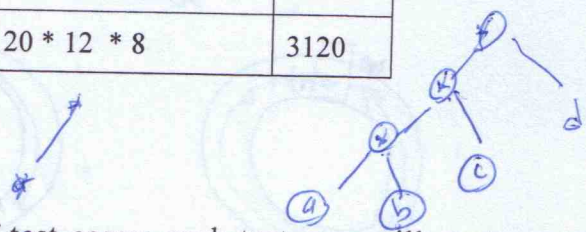
Matrix	Dimension
A	20 x 2
B	2 x 30
C	30 x 12
D	12 x 8

Table 1

	Order of multiplication	Computations	Total
1	A(B(CD))	30 * 12 * 8 + 2 * 30 * 8 + 20 * 2 * 8	3680
2	(AB) (CD)	20 * 2 * 30 + 30 * 12 * 8 + 20 * 30 * 8	8880
3	***A((BC)D)***	***2 * 30 * 12 + 2 * 12 * 8 + 20 * 2 * 8***	***1232***

4	((AB)C)D	20 * 2 * 30 + 20 * 30 * 12 + 20 * 12 * 8	10320
5	(A(BC))D	2 * 30 * 12 + 20 * 2 * 12 + 20 * 12 * 8	3120

Table 2



Input

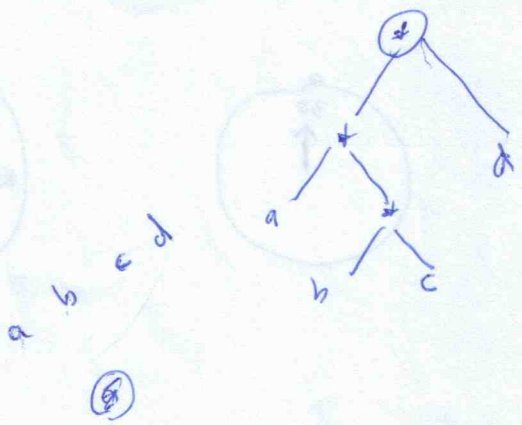
The first line of the input indicates the number of test cases; each test case will start with an integer n (n<1000) for the number of matrices followed by n lines for each matrix. For each matrix, this line contains the dimensions (number of rows followed by number of columns).

Output

For each test case, output the number of multiplications needed for the optional order.

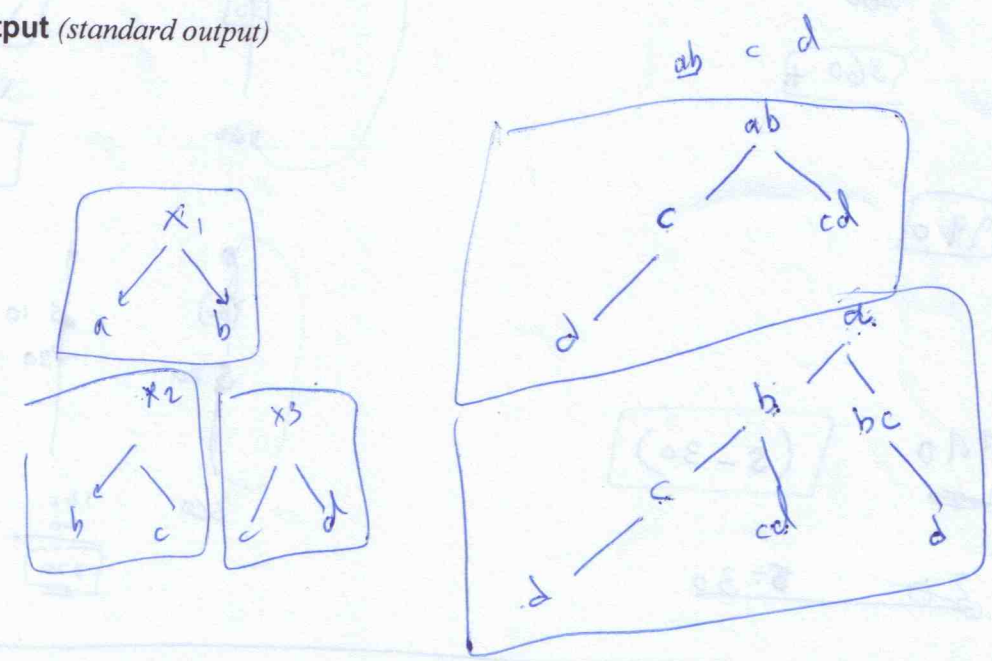
Sample Input *(matrix.in)*

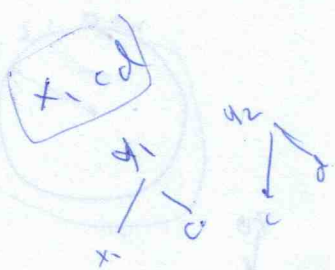
```
2
3
3 2
2 4
4 2
4
20 2
2 30
30 12
12 8
```

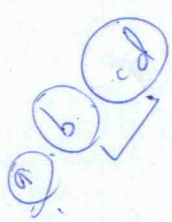


Sample Output *(standard output)*

```
28
1232
```





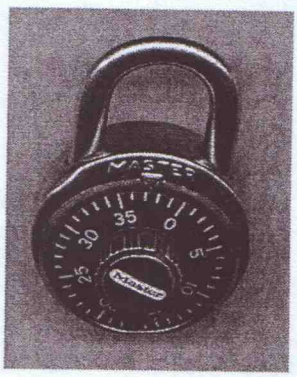


Problem H:

Combination Lock

Source file: lock {.c| .cpp| .java}
Input file: lock.in

A common combination lock design is shown in the picture below. The lock has a dial with 40 calibration marks numbered 0 to 39. A combination is composed of three of these numbers such as 10, 2 and 18. Assuming that the lock is at an initial position (i.e. a number s where 0<=s<=39), the user need to do the following steps to open the lock:



- Turn the dial clockwise 2 full turns (rotations) from the initial position.
- In the same direction, keep going until you reach the first number of the combination.
- Turn the dial counter-clockwise one full turn from the actual position (the first number of the combination) and continue turning counter-clockwise until the 2nd number is reached.
- Turn the dial clockwise again until the 3rd number is reached; the lock will now be open.

Notice that the pointer indicating the position does not move but rather the dial (the circular bar containing the numbers from 0 to 39) is the one which moves. For example, if the starting position was 0 and you rotate the dial clockwise or counter clockwise until 0 become again the initial position, you would have completed a full turn of 40 calibration marks equivalent to 360 degrees. If you move the dial 5 calibration marks clockwise from position 0 it will be positioned at 35 and if you move the dial 5 calibration marks counter-clockwise from position 0 it will be positioned at 5 and so on.

Given a combination for the lock, and the number of degrees the dial was rotated in total (clockwise plus counter-clockwise) to open, discover the initial position of the lock.

Input

- Input consists of several test cases.
- For each case there is a line of input containing 4 numbers separated by a single space:
  - The fist number is the total number of degrees (clockwise plus counter-clockwise) a lock was rotated.
  - The following three numbers represent the combination; consecutive numbers in any combination will be distinct.
  - A line containing four zeros (0 0 0 0) follows the last case.
- Example: the input **1350 30 0 30** and the output **0** indicates that it was necessary to rotate the lock 1350 degrees to open using the combination 30, 0 and 30 from the initial position 0.

Output

For each case, print a line with a single integer: the initial position of the dial.

Sample Input *(lock.in)*

```
1350 30 0 30
1620 20 0 20
1620 27 7 27
1890 10 0 10
1197 11 19 15
0 0 0 0
```

Output for Sample Input *(standard output)*

```
0
0
7
0
12
```

Problem I:

A Voyager

Source file: voyager {.c| .cpp| .java}
Input file: voyager.in

A voyager sets out to explore a new area in a city that he is visiting. Being cautious, he takes his mobile phone with him, so he can call for help in case of an emergency.

The mobile phone works as long as it is in range of at least one phone transmission tower in the area. All transmission towers have the same area of coverage and every area of coverage is a circular area. The voyager has a map of the area showing the locations of the phone transmission towers.

The voyager knows that the best path from the source to destination will only change directions on intersection points between circles (representing the areas of coverage) and only on the boundary of the reachable terrain. Your task is to help the voyager to take the shortest route, while staying in range of at least one of the towers.

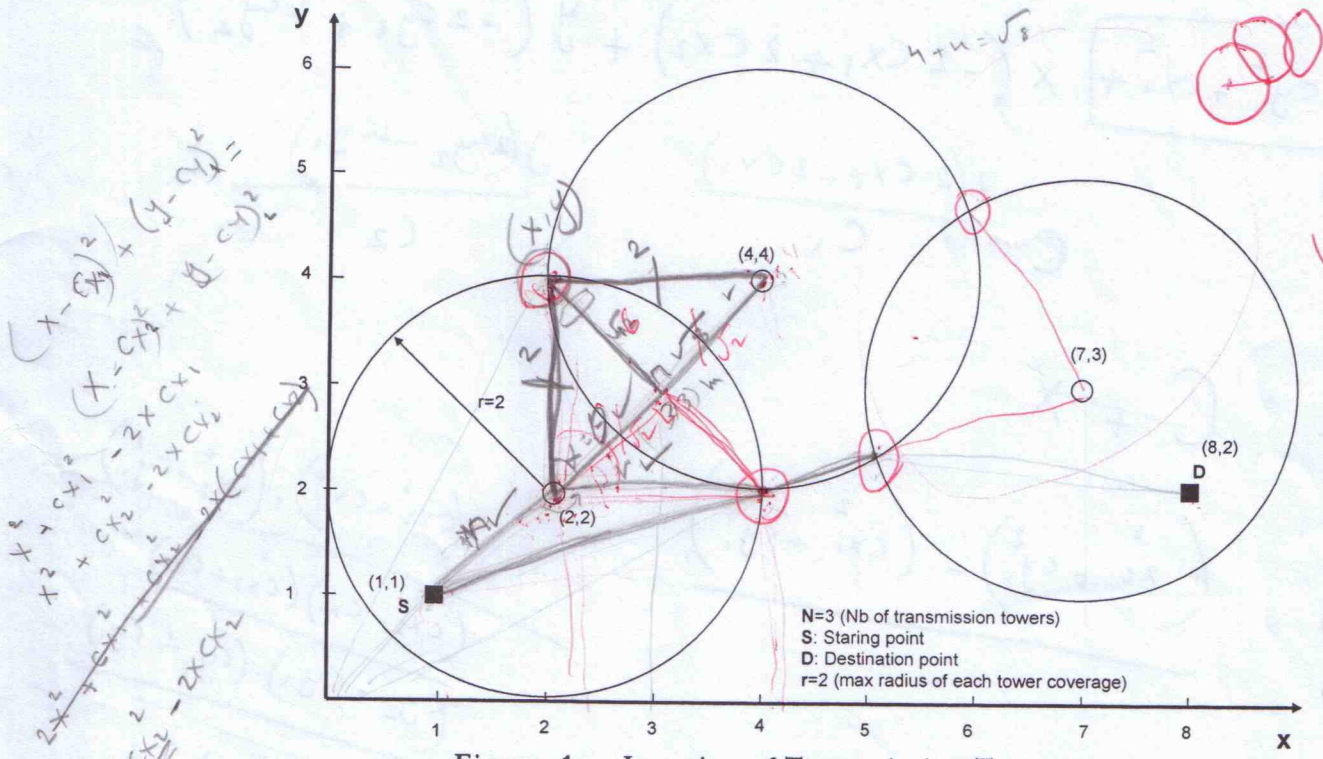


Figure: 1a – Location of Transmission Towers

Input

- The first line is an integer c ($1 \le c \le 99$) which represents the number of test cases.
- Each test case consists of:
  - A line containing two integers: r ($1 \le r \le 999$) and N ($1 \le N \le 99$) where r is the radius of the area covered by a transmission tower and N is the number of transmission towers on the map.
  - One line indicate the x, y coordinates of the starting point **S** on the map.

- One line, indicates the x, y coordinates of the destination point **D** on the map.
- **N** lines where each line indicates the x, y coordinates of the tower location.

Notes

- A location is specified by two space-separated integer coordinates, x and y ($1 \leq x, y \leq 999$).
- Locations in the input for each single test case will be distinct. The starting locating **S** will not be more than r units away from the nearest tower.

Output

For each test case: One value which is the distance to the goal location, or −1 if the goal location can't be reached. Your answer should have either an absolute or a relative error of at most 10^{-4}.

Note:

An example scenario is given in *Figure: 1a – Location of Transmission Towers*, based on the Sample Input given below.

Sample Input *(voyager.in)*

```
2
2 3
1 1
8 2
2 2
4 4
7 3
2 3
1 1
8 2
2 2
4 4
7 4
```

Sample Output *(standard output)*

```
7.23224071072994
-1
```