Problem A

Quadtrees

Input: standard input
Output: standard output

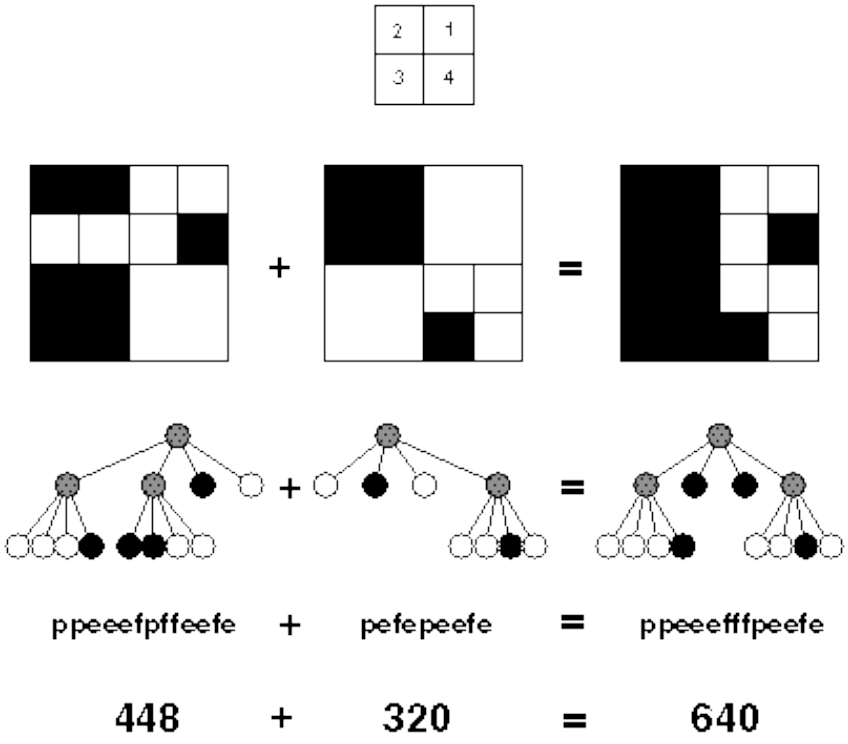
A quadtree is a representation format used to encode images. The fundamental idea behind the quadtree is that any image can be split into four quadrants. Each quadrant may again be split in four sub quadrants, etc. In the quadtree, the image is represented by a parent node, while the four quadrants are represented by four child nodes, in a predetermined order.

Of course, if the whole image is a single color, it can be represented by a quadtree consisting of a single node. In general, a quadrant needs only to be subdivided if it consists of pixels of different colors. As a result, the quadtree need not be of uniform depth.

A modern computer artist works with black-and-white images of 32×32 units, for a total of 1024 pixels per image. One of the operations he performs is adding two images together, to form a new image. In the resulting image a pixel is black if it was black in at least one of the component images, otherwise it is white.

This particular artist believes in what he calls the *preferred fullness*: for an image to be interesting (i.e. to sell for big bucks) the most important property is the number of filled (black) pixels in the image. So, before adding two images together, he would like to know how many pixels will be black in the resulting image. Your job is to write a program that, given the quadtree representation of two images, calculates the number of pixels that are black in the image, which is the result of adding the two images together.

In the figure, the first example is shown (from top to bottom) as image, quadtree, pre-order string (defined below) and number of pixels. The quadrant numbering is shown at the top of the figure.



Input:

The first line of input specifies the number of test cases (N) your program has to process. The input for each test case is two strings, each string on its own line. The string is the pre-order representation of a quadtree, in which the letter 'p' indicates a parent node, the letter 'f' (full) a black quadrant and the letter 'e' (empty) a white quadrant. It is guaranteed that each string represents a valid quadtree, while the depth of the tree is not more than 5 (because each pixel has only one color).

Output:

For each test case, print on one line the text 'There are X black pixels.', where X is the number of black pixels in the resulting image.

Sample Input:

3
ppeeefpffeefe
pefepeefe
peeef

peefe
peeef
peepefefe

Sample Output:

There are 640 black pixels.
There are 512 black pixels.
There are 384 black pixels.

Problem B

Finding an inverse of a square matrix

Input: standard input
Output: standard output

Consider a square matrix A with n*n elements, where $2 \leq n$. You can key in n*n elements separated by space. If A is invertible, then the inverse matrix A^{-1} of order n will be computed and displayed. If A^{-1} does not exist, then the message “singular” will be displayed.

Input:

The input consists of several test cases. First line of each case is case title. Following each line is mapped to each row of matrix A. A is a n*n matrix. The number n can be determined automatically by program. Each row is separated by the “Enter” of keyboard. Each element of the same row is separated by one or more blank characters.

Output:

For each test case, display the inversed matrix A^{-1}. If the element is not integer, two digits of fractional part are inclusive.
Follow the format of the sample output.

Sample Input:

Case 1:
5 -3
-1 1

Case 2:
1 -2.00 1
-2 1 3.00
-1 -4 9

Case 3:
1 1 1 2

1 2 1 3
0 0 0 -1
0 0 -1 0

Sample Output:

Case 1:
0.50 1.50
0.50 2.50

Case 2:
Singular

Case 3:
2 -1 1 1
-1 1 1 0
0 0 0 -1
0 0 -1 0

Problem C

Variable-Length Encoded Number

Input: standard input
Output: standard output

For safety reason, positive integers in the input file are encoded based on the following table. Each line in the input file consists of one and only one integer. Each integer will be transformed to a string of variable length. Digits in an integer will be transformed one by one, from left to right, and no space is between any encoded consecutive strings. For instance, 256 will be transformed into pearplumgrape.

Please write a program to read all the positive integers from the input file, add them up, and print the sum on the standard output.

Digit	Encoded String
0	apple
1	banana
2	pear
3	melon
4	mango
5	plum
6	grape
7	lemon
8	berry
9	cherry

Input:

The input consists of several test cases. First line of each case has a number n. The following n line, each line has one variable-length string. No space is contained in any string.

Output:

The sum of all the encoded positive integers in the input.

Sample Input:

3
mango
plumberryberry
cherrylemon

4
bananaapplepearmelon
pear
meloncherrypear
cherryapple

Sample Output:

689
1507

Problem D

Formatted String Expression

Input: standard input
Output: standard output

A string can always be described as some formats of expressions. For example, abaa, ababaab, aaababababaabb, and aabababababb can be expressed as the format of $a^{n}((ba)^{*}baa)^{n}b^{*}$, where

1. only a and b characters will appear in this expression.
2. n describes that the associative character/characters must appear once or more times.
3. * describes that the associative character/characters must appear any number of times, including zero.
4. n and * in different position may represent different number of associative character/characters.

Therefore, abaabb can be expressed as $a^{1}((ba)^{0}baa)^{1}b^{2}$,
ababaa can be expressed as $a^{1}((ba)^{1}baa)^{1}b^{0}$,
aaababababaabb can be expressed as $a^{3}((ba)^{2}baa)^{1}b^{2}$,
aabababaababababaabb can be expressed as $a^{2}((ba)^{2}baa)^{2}b^{2}$.

Please write a program to check how to express a string A as the format B.
The superscript characters will be replaced with normal style characters in input file. Thus "$a^{n}((ba)^{*}baa)^{n}b^{*}$ " will be express as "an((ba)*baa)nb*" in input file.

Input:

The first line will be a testing string A.
The second line will be the given format B.
If there are additional cases need to be tested, they will be shown on successive lines with the same definitions as above.

Output:

If a testing string A can not be expressed as the given format B, please print “The input string can not be expressed as given format.” message.

If an input string A can be expressed as the format B, please print “The input string “ A ” can be expressed as “B”; where A is the testing string and B is the given format in input file. You need to specify the precise number for n and * in the format B.

For example, if the input string is aabaaba and the given format is an(ab)*an(aba)*, the output will be: The input string aabaaaba can be expressed as a1(ab)1a2(aba)1.

Sample Input:

aaababababbbba
an(ba)*bn
aababababaababaabb
an((ba)*baa)nb*

Sample Output:

The input string can not be expressed as given format.
The input string aababababaababaabb can be expressed as a2((ba)2baa)2b2

Problem E

The link expiration time (LET) between two vehicles

Input: standard input
Output: standard output

In vehicular ad hoc network (VANET), the link expiration time (LET) is used to represent the maximal lifetime of the wireless link, i.e., the maximal time duration to transfer packets, between two vehicles. The motion estimation technique is proposed to calculate the LET of two neighboring vehicles in VANET. Assume clocks of all vehicles in VANET are synchronized. Two-dimensional coordinates of vehicles i and j at time t are denoted as (x_i, y_i) and (x_j, y_j), respectively. Further, θ_i and θ_j $(0 \le \theta_i, \theta_j < 2\pi)$ are their moving directions and v_i and v_j are their moving speeds. If r is the maximal communication range of the wireless link between two vehicles, the following equation can be used to calculate the LET of vehicles i and j.

$$LET_{ij} = \frac{-(ab+cd)+\sqrt{(a^2+c^2)r^2-(ad-bc)^2}}{a^2+c^2}$$

where

$a = v_i \cos\theta_i - v_j \cos\theta_j$

$b = x_i - x_j$

$c = v_i \sin\theta_i - v_j \sin\theta_j$

$d = y_i - y_j$

Please note that the symbol *INF* is used to represent the value of LET_{ij} when $v_i = v_j$ and $\theta_i = \theta_j$.

Input:

The first line will be the given r. Each following line will be the given (x, y, v, θ) of each vehicle, where $0 \le \theta < 360$.

Output:

You need to print out the result of every testing case with the following format to the fourth decimal with the rest-digit rounding. Please start the output from $i = 1$.

LET_{ij} = the LET value of vehicles i and j, where $1 \leq i < j$.

Sample Input:

2
0 0 1 0
0 1 1 90
-1 0 1 180

Sample Output:

LET(1,2)=0.8229
LET(1,3)=0.5000
LET(2,3)=0.4142

Problem F

Pattern Recognition and Accumulation

Input: standard input
Output: standard output

A special pattern character string is formatted as XYX (1. The length of the string is three. 2. The first and third characters are identical). Given the target pattern and a long character string (less than 1000 characters), write a program that can recognize the target pattern and calculate how many times it shows up in the long character string. Overlapped patterns are allowed and should be all counted. In other words, XLBLBLBLY contains not just 2 but 3 LBL patterns

Input:

A target pattern (XYX formatted) with length 3 and a long character string with length no more than 1000.

Output:

The number of target pattern (overlapped patterns should be all counted) found in that long string. If the input is invalid (say pattern length is not 3, pattern is not XYX formatted, or the long string is longer than 1000), return “Invalid input format“.

Sample Input:

LPL BEGINTHELLPPLPLXYXYXYPLPLPLPLLLPLEND
GGG BEGINGGYGGGGGHKGGGGGGGGYGEND
LKK BEGINLKKLKKKLKEND
LKLK BEGINLKLKLKKPPOOGGYYLKLKLKLEND

Sample Output:

6
9
Invalid input format
Invalid input format

Problem G

Up and Down Sequences

Input: standard input
Output: standard output

The quality of pseudo random-number generators used in some computations, especially simulation, is a significant issue. Proposed generation algorithms are subjected to many tests to establish their quality. One of the common tests is the *run test*.

In this test, sequences are tested for “runs up” and “runs down.” We will examine series of data values for the “Up” and “Down” sequences each series contains. Within a series, an “Up” sequence continues as long as each data-value received is not less than the previous data-value. An “Up” sequence terminates when a data-value received is less than the previous data-value received. A “Down” sequence continues as long as each data-value received is not greater than the previous data-value. A “Down” sequence terminates when a data-value received is greater than the previous data-value received.

An “Up” sequence can be initiated by the termination of a “Down” sequence and vice versa. (Sequences initiated in this manner have length one at this initiation point.)

All the initial data-values are part of an “Up” sequence, and contribute to its length, if the first deviation of the data-values is upwards. All the initial data-values are part of a “Down” sequence, and contribute to its length, if the first deviation of the data-values is downwards. If the data-values received don’t allow classification as either an “Up” or a “Down” sequence, the data should be considered to have neither sequence.

Find the average length of both the “Up” and the “Down” sequences encountered for each input line in the data file. Report these average lengths as each input line is processed.

Input:

Each of the separate series to be examined is contained on a single line of input.

Each series to be analyzed consists of at least one and no more than 30 unsigned, non-zero integers.

Each integer in a series has at least one digit and no more than four digits. The integers are separated from each other by a single blank character.

Each of the series will be terminated by a single zero (0) digit. This terminator should not be considered as being part of the series being analyzed.
The set of series to be analyzed is terminated by a single zero (0) digit as the input on a line. This terminator should not be considered to be a series, and no output should be produced in response to its encounter.

Output:

A line with two real values is to be emitted for each input data set encountered. It must begin with the message “Nr values = N: ”, where N is the number of input data in the line and then to continue with the average values for runs. First, the average “Up” run length, then the average “Down” run length. Separate these values with a space. Answers must be rounded to six digits after the decimal point

Sample Input:

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

Sample Output:

Nr values = 3: 2.000000 0.000000
Nr values = 3: 0.000000 2.000000
Nr values = 5: 2.000000 2.000000
Nr values = 5: 4.000000 0.000000
Nr values = 5: 0.000000 4.000000
Nr values = 7: 0.000000 6.000000
Nr values = 7: 1.000000 5.000000
Nr values = 4: 0.000000 0.000000
Nr values = 7: 2.500000 1.000000

Problem H

Ship Location Guessing

Input: standard input
Output: standard output

Probably everyone who ever attended school knows the game where two opposing players place a set of ships on a sheet of paper and try to eliminate each other's ships by guessing their location. In our version of the game, your opponent has distributed the following **seven** ship patterns over a rectangular grid of squares:

```
xx   xx     xx   x       x    x
xx    xx   xx    xxx   xxx   xxx   xxxx
```

Each ship pattern covers exactly four squares. The patterns may be **rotated** but **not mirrored**. All patterns are guaranteed to be placed completely within the boundaries of the rectangle and not to overlap each other, whereas touching another pattern or the border is allowed.

We assume that we are in the middle of the game and that several squares have already been uncovered. You will be given a rectangular grid of squares representing your current knowledge about the positions of your enemy's ships. Every square is marked by one of the following characters:

- 'x' if a ship covers the square
- 'o' if no ship covers the square
- 'u' if the square has not yet been uncovered

Given that information, you are to decide whether you can determine all remaining 'x' squares with at most one miss, i.e. whether you could uncover the 'u' squares without getting more than one 'o' square before you had all 'x' squares uncovered. This means you are allowed to hit a 'o' if then the solution becomes unique.

Input:

The input file contains several game situations. Every test case starts with a line containing two integers h and w. These define width and height of the game rectangle, where 2 <= w, h <= 16. Each of the next h lines contains a string of w characters.

Each of these characters is either `x', `o' or `u', depending on the state of the corresponding square. A blank line separates each game from the next. The input file ends with a game having w = 0 and h = 0. This game should not be processed.

Output:

For each test case you should first output a line containing the number of the game, followed by a line containing either `yes.' (if you can determine all `x' with at most one miss) or `no.' (if you cannot determine all `x' without at least two misses). Output a blank line after every game.

Sample Input:

10 10
uxuuxuuuuu
ooooxoooo
oxooxxxuuu
xxooooooouu
xoooxooouu
ooxxxxooouu
oooooxxoox
ooooooxoox
ooooooooxx
oooooooooo

0 0

Sample Output:

Game #1
yes.

Problem I

Two-Dimensional Cellular Automata

Input: standard input
Output: standard output

Given a two-dimensional 5*5 array of cells in which each cell is either dead or alive, the values for the next generation of cells are based on the following rules.

- A dead cell is marked as 0, while a living cell is marked as 1. Each cell is adjacent to 3-8 neighbor cells (see Figure 1). For example, the living cell (1) with row=2 and column=3 has eight neighbors, and four of them are living cells. The dead cell (0) with row=2 and column=1 has five neighbors, and three of them are living cells.

0	1	0	0	1
0	1	1	1	1
0	1	0	0	1
0	0	0	1	0
0	0	0	0	0

Figure 1. An example of a five-by-five array of cells

- In the next generation, any living cell with only 0 or 1 living neighbors will die from isolation. Any living cell with 4 or more living neighbors will die from overcrowding. Any living cell that is adjacent to exactly 2 or 3 other living cells remains unchanged (i.e. continues to live).
- In the next generation, any dead cell with exactly 3 living neighbors will become alive. Any dead cell that is adjacent to any less than or greater than 3 other living cells remains unchanged (i.e. still be a dead cell).
- Figure 2 shows the next generation of the cell array in Figure 1. For example, the dead cell with row=2 and column=1 becomes alive in Generation 2, since this cell has 3 living neighbors. On the contrary, the living cell with row=2 and column=3 becomes dead in Generation 2, since this cell has 4 living neighbors.

0	1	0	0	1
1	1	0	0	1
0	1	0	0	1
0	0	0	0	0
0	0	0	0	0

Figure 2. The next generation of the cell array in Figure 1

Input:

The input consists of several test cases. First line of each case has a number n. The following n case contains a 5*5 array of cells in Generation 1. There are five lines in which each line has five cells. Cells are separated by spaces. A blank line separates each case.

Output:

For each test case, output five generations of cell arrays (including Generation 1). Each generation should output a 5*5 array of cells in which cells are separated by spaces.

Sample Input:

2

0 1 0 0 1

0 1 1 1 1

0 1 0 0 1

0 0 0 1 0

0 0 0 0 0

1 1 1 0 0

0 1 0 1 0

0 1 0 1 1

1 0 1 1 0

0 1 0 0 0

Sample Output:

#Case 1:

Generation 1:

0 1 0 0 1

0 1 1 1 1

0 1 0 0 1

0 0 0 1 0

0 0 0 0 0

Generation 2:

0 1 0 0 1

1 1 0 0 1

0 1 0 0 1

0 0 0 0 0

0 0 0 0 0

Generation 3:

1 1 0 0 0

1 1 1 1 1

1 1 0 0 0

0 0 0 0 0

0 0 0 0 0

Generation 4:

1 0 0 1 0

0 0 0 1 0

1 0 0 1 0

0 0 0 0 0

0 0 0 0 0

Generation 5:

0 0 0 0 0

0 0 1 1 1

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

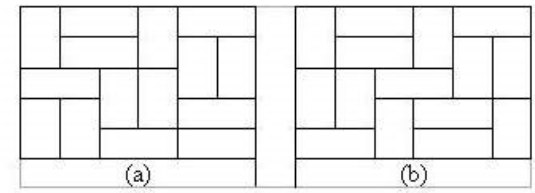
#Case 2:
Generation 1:
1 1 1 0 0
0 1 0 1 0
0 1 0 1 1
1 0 1 1 0
0 1 0 0 0
Generation 2:
1 1 1 0 0
0 0 0 1 1
1 1 0 0 1
1 0 0 1 1
0 1 1 0 0
Generation 3:
0 1 1 1 0
0 0 0 1 1
1 1 1 0 0
1 0 0 1 1
0 1 1 1 0
Generation 4:
0 0 1 1 1
1 0 0 0 1
1 1 1 0 0
1 0 0 0 1
0 1 1 1 1
Generation 5:
0 0 0 1 1
1 0 0 0 1
1 0 0 1 0
1 0 0 0 1
0 1 1 1 1

Problem J

Unbreakable Floor

Input: standard input
Output: standard output

Alan bought a new house. He likes rectangles, so he wants his floor full of identical rectangular shapes. Imagine he has a floor of **5 x 6**, he may fill this floor with rectangles of **1 x 2** in at least two ways:



Picture (a) shows a 'breakable' layout, since there is a straight line through the whole floor which divides the floor into two parts -- a **5 x 4** rectangle and a **5 x 2** rectangle, and all the **1 x 2** rectangles are not destroyed.
Picture (b) shows a 'unbreakable' layout, since you cannot divide it into two parts without destroying any **1 x 2** rectangle.
Alan likes unbreakable floorings, but he's not sure if it is possible for any size of floor and rectangle shape. Can you tell him?

Input:

The first line contains the number of tests **t(1<=t<=40)**. Each case consists of a single line with four positive integers **p, q, a, b (1<=p, q, a, b<=10000)**.

Output:

For each test case, print the case number first. Then print the word '**Yes**' if it is possible to make a unbreakable floor of **a x b** with rectangles of **p x q**, otherwise print the word '**No**'. Answer for each case should be in exactly one line.

Sample Input:

2
1 2 5 6

1 2 3 17

Sample Output:

```
Case 1:Yes
Case 2:No
```