Problem 1: Cowculations (D. T. Joyce, Villanova U., 1996)

A primitive cow culture was discovered by noted anthropologist Dr. Bo Vine. Hundreds of computation tablets were unearthed in a pasture somewhere near Dallas. Dr. Vine managed to decipher the mystery of the tablets when he realized they represented mathematical calculations.He says "I've always suspected that cows are smarter than they lead us to believe and here's the proof. The big breakthrough was realizing that they are not able to do math by cownting on their fingers, but they are able to think on their feet. But now I have hundreds of these tablets and I need help in verifying my hypothesis."

Problem Statement: Write a program that helps Dr. Vine verify his hypothesis. This problem statement describes how he thinks the tablets should be interpreted. Each tablet contains six lines and each line contains a sequence of cow symbols. The first two lines represent cow numbers, the next three represent operations on those numbers and the sixth represents the result. Cow numbers make use of four symbols: V U C and D most closely represent the marks made by the cow hooves, so we will use them to represent the symbols. The numbers on the first two lines of a tablet are always a sequence of 5 of these symbols, and the number on the sixth line of a tablet is always a sequence of 8 of these symbols.

To interpret the cow calculations one must keep track of two numbers, hereafter referred to as Num1 and Num2. Originally Num1 is equal to the number on line 1 of a tablet and Num2 is equal to the number on line 2 of a tablet. However the value of Num2 can change as a result of the operations. There are four possible operations, represented by the symbols A, R, L, and N. Operation A' causes Num2 to become the "sum" of Num1 and Num2, using this addition table:

A	V	U	C	D
V	V	U	C	D
U	U	C	D	V,U
C	C	D	V,U	U,U
D	D	V,U	U,U	C,U

The first symbol in a result box represents the result of the addition. The second symbol in a box represents a carry symbol.
For example U A' V = U and C A' C = V with a carry of U.

Examples of cow addition are:

VUCDV A' VUCDV = VDUCV and DVVCU A' CVUCU = UUVCVC.

Operation R causes Num2 to have its symbols shifted right one position, with the rightmost symbol being lost and a V' being placed into the leftmost position. For example VVCDU would become VVVCD. Operation L causes Num2 to have its symbols shifted left one position, with the leftmost symbol being kept and a V' placed into the rightmost position. For example VVCDU would become VVCDUV. N is the null operation. It has no effect on Num2. After the operations have been performed on Num1 and Num2 the final value of Num2 is hopefully the value marked on the sixth line of the tablet. If the final value of Num2 does not make use of 8 symbols then it is padded with V's on the left (VDCCC becomes VVVVDCCC).

Dr. Vine says "I have already verified that all the tablets conform to the hypothesis in terms of the format of the tablets, number of lines and symbols, types of symbols, etc. But I need to verify that the result of the mathematical operation described on lines 1 through 5 is indeed on line 6."

Input: The first line contains an integer N between 1 and 10 describing how many tablets are represented. The next 6 * N lines represent the N tablets as described in the problem statement.

Output: There should be N+2 lines of output. The first line of output should read COWCULATIONS OUTPUT. There will then be one line of output for each tablet that states either "Yes" (the tablet follows Dr. Vine's hypothesis) or "No" (the tablet does not follow Dr. Vine's hypothesis). The final line of output should read "END OF OUTPUT".

Example: The following input data:

```
5
VVVVU
VVVVU
A
A
A
VVVVVVUV
VVCCV
VVDCC
L
R
A
VVVVUCVC
VVCCV
VVDCC
R
L
A
VVVVUCVV
VVUUU
VVVVU
A
N
N
VVVVVUCU
DDDDD
VVVVU
A
L
L
UVVVVVVV
```

should produce the following output:

```
COWCULATIONS OUTPUT
YES
YES
YES
NO
YES
END OF OUTPUT
```

Problem 2: Pseudoku (O. Kaser, UNB Saint John, 2008)

In the game of Pseudoku, a 4x4 grid is to be filled in with the letters 'a', 'p', 'i', 'c' and 's'. The pseudoku grid is divided into 4 quadrants (upper left, upper right, lower left and lower right). Each quadrant is 2 x 2. The grid is presented to the user with some of the letters already filled in, and the player is supposed to fill in the remainder, subject to some conditions:

1. In any horizontal line, there cannot be multiple occurrences of any character.
2. In any vertical line, there cannot be multiple occurrences of any character.
3. In any quadrant, there cannot be multiple occurrences of any character.

You may assume that the inputs provided to you do not initially violate the 3 conditions. However, it is not always possible to complete the grid.

In many cases, there may be several ways to fill in the rest of the grid, so we impose an additional rule: The correct solution is lexicographically first, when the grid is read like a page (read the top line first, and each line is read left to right). In other words, if the grid can be completed in several ways, we are looking for the way that would be first, if the different solutions were put in alphabetical order.

INPUT:
Several partially completed pseudoku grids.

The format is as follows:
First, there is a single positive integer indicating the number of grids (say N) to follow. Then, there are 5N lines, where each grid occupies 5 lines (four lines with 4 characters each, followed by a blank line). Blanks are indicated by the dash character (-) in the input.

OUTPUT:
For each input grid, the lexicographically first solution, if there is a solution. Otherwise, the string "no solution".

The required format is as follows:
For each input grid that has a solution, the completed results (each - character has been replaced by 'a', 'p', 'i', 'c' or 's'). This will occupy 4 lines, each with 4 characters. A blank line will follow the solution. For each input grid that has no solution, the string "no solution", followed by a blank line.

EXAMPLE:
Input:

```
3
apic
siap
ca--
p---

-api
c---
s---
----
```

```
a---
-p--
--i-
---c
```

****

Output:

```
apic
siap
capi
psca

no solution

acpi
ipas
caip
pisc
```

****

Note: in the last case, another way of filling in the grid is

```
aicp
cpai
pais
ispc
```

However, the right answer is lexicographically smaller than this.

Problem 3: Roundabout (T. Wareham, MUN, 2008)

A laboratory robotic system consists of a circular 360º reagent rack surrounding a central analytic station with attached robotic arm and item basket. Each reagent has a degree-position on the rack where it is located and a particular weight. The item basket has a fixed weight limit L. A trip by the robot arm consists of an initial move out to a particular reagent at degree-position p, 1 <= p <= 360, and the placing of that reagent in the basket, the move to and grabbing of subsequent reagents going to the right around the rack of a reagents (by increasing degree and possibly wrapping around after 360º), and the final return to the central analytic station. During a trip, the summed weight of the accumulated reagents in the basket cannot exceed L. The requirements for a particular analysis consists of a sequence of n degree-position / weight pairs corresponding to the required reagents in ascending order by degree-position. Reagents must be picked up in the specified sequence-order (possibly wrapping around after 360o), but any reagent may be the first picked up in the first trip. For example, given the requirements {(10, 20), (100, 5), (105, 10), (240, 5)} and L = 25, two possible trip-sequences are {[(100, 5)], [(105, 10), (240, 5)], [(10, 20)]} and {[(105, 10), (240, 5)], [(10, 20), (100, 5)]} requiring 3 and 2 trips, respectively, with the latter requiring the minimum number of trips for the given requirements relative to L.

Write a program which, given the requirements for an analysis and L, computes the minimum number of trips required to bring all required reagents to the central analytic station. The input consists of a file in which the first line specifies the number of test cases and each n-reagent requirements test case is described by an (n+ 1)-line block in which the first line specifies n and L and the remaining lines in the block specify the degree-position and weight of each reagent. Each block (including the first line of the file) is followed by a blank line to make the file easier to read. You may assume that all input files are formatted correctly and that no reagent's weight exceeds L.

Sample input:

```
  5

  4   25
 10   20
100    5
105   10
240    5

  5   15
 10    5
 40   15
 45    5
 90   10
100    5

  5   20
 10    5
 40   15
 45    5
 90   10
100    5
```

```
  5   40
 10    5
 40   15
 45    5
 90   10
100    5

  8   50
 45   25
 90   30
135   15
180   30
225   20
270   50
315   10
360   10
```

Sample output:

```
Case #1: Minimum number of trips = 2
Case #2: Minimum number of trips = 3
Case #3: Minimum number of trips = 2
Case #4: Minimum number of trips = 1
Case #5: Minimum number of trips = 4
```

Problem 4: Help me read Paul's messages (J. Almhana/E. Hervet, U. de Moncton, 2008)

Paul is sending coded messages to his friends. He coded his messages as follows:
The letters of each word of x characters are right rotated by y mod x, where y is computed from another coding procedure described below. We suppose that all words contain capital letters only and no line has more than 999 characters not counting the end-of-line. All other characters (spaces, commas, periods, etc.) that can be used as delimiters between words, are not altered, and are left at their current position.

Computing y

Let's consider the following coding table:

Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Corresponding code	J	S	B	G	T	O	C	X	P	Q	R	I	F	D	A	U	Y	N	V	H	L	K	W	Z	M	E

Step 1: Finding the number of iteration n for a coding cycle.
Using the above table, the corresponding code for the word ANIMAL is JDPFJI. We can apply the coding process again on the word JDPFJI to obtain QGUOQP which can lead to another word YCLAYU using the same procedure. We repeat this step a number of times n that leads back to the original word ANIMAL. In this particular case, n equals 252.

ANIMAL → JDPFJI → QGUOQP → YCLAYU→……→ ANIMAL

Step 2: Computing the numeric root y of n.
The numeric root of a number *x* is computed as follows:

a- y = sum of all x's digits,
b- if y > 10 then x=y, go to step a
else y is the numeric root.

Input:
The input file contains the following:
Line 1: the alphabet used in Paul's messages.
Line 2: the corresponding code.
Following lines: Paul's coded message.

Output:
Paul's decoded message.

Example of input file:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
JSBGTOCXPQRIFDAUYNVHLKWZME
HET MPETITIONCO SI LLYREA OUGHT!
HOW SI ONNAG WIN STHI RYEA?
```

Corresponding output:

```
THE COMPETITION IS REALLY TOUGH!
WHO IS GONNA WIN THIS YEAR?
```

Problem 5: 2D moving company (A. Sedgwick, Dalhousie U., 2008)

Have you ever had the following experience of trying to move a large sofa down some stairs around sharp corners and discovering it could not be done. This problem is a much-simplified two-dimensional version of the problem above. You have some rectangular boxes which need to be moved through various corridors and around some right angle turns. You are not allowed to tilt them but may rotate them.

The input to this problem consists of two lines. The data on the first one describes the corridors and turns. The second line contains the width and length of each box. The width may be less than or greater than the length.

The one line of output should contain the data for those boxes which could successfully navigate the doorways and corridors and turns in the same order they appear in the input and with the width and length as in the input. However, the number of spaces between numbers may be different. The input may be free-format but the output should have two spaces between boxes and one space between each width and length.

You may assume that the input is sensible and all numbers are positive. You may also assume that the corridors are long enough and that there is plenty of room at the start and end to orient the boxes as necessary.

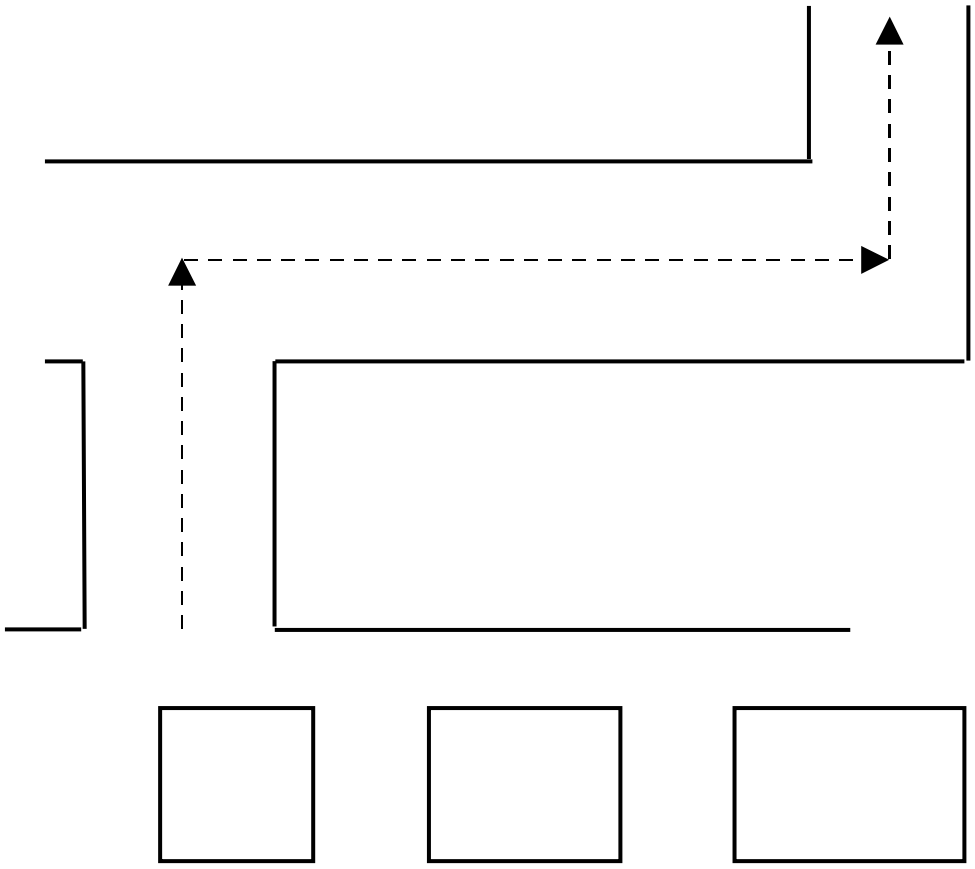
Sample input:

```
5 R   5 L   4 E
4 4   5 4   6 4
```

The first item of each pair is the width and the second is the direction to turn (L for left, R for right, or E indicating the end of input. Turns occur at the end of a corridor. It is not always necessary to rotate a box at a turn. (Sometimes you push the box through and then start pushing the box at a right angle to the direction it was previously going.)

Corresponding output:

```
4 4  5 4
```



Note that the first box will just fit in the third corridor. The second box needs to go sideways down the second corridor (with long side leading) in order make the second turn (without rotating). The third box can just make the first turn but not the second turn.

You may use the `sqrt()` function (but it is not necessary).

Problem 6: Longest subsequence (J. Almhana/Z. Liu, U. de Moncton, 2008)

In mathematics, a subsequence of a sequence is a new sequence which is formed from the original sequence by deleting some of the elements without disturbing the relative positions of the remaining elements. If the elements of a subsequence are in non decreasing order, the subsequence is called a non decreasing subsequence.

Formally we can express the above definition as follows:
Let X = {x[1], x[2], …, x[n]} be a sequence of n non-negative integers and $\{i_1,i_2,\ldots,i_k\}$ be a subset of {1,2,3,…, n} satisfying $i_1<i_2<\ldots<i_k$, then $\{x[i_1], x[i_2], \ldots, x[i_k]\}$ is called a subsequence of X. Subsequence $\{x[i_1], x[i_2], \ldots, x[i_k]\}$ is called **non-decreasing** if $x[i_1] \leq x[i_2] \leq \ldots \leq x[i_k]$.

For example:
X = {4, 3, 6, 8, 7, 5} is a sequence of 6 integers.
X1 = {3, 5} and X2 = {6, 8} are two non-decreasing subsequences of length 2.
X3 = {3, 6, 8} and X4={4, 6, 8} are non-decreasing subsequence of length 3.

Write a program to find the length of the longest non decreasing subsequence of X.

Example of input:

```
96  62   6   2  84  20  50  73  80  14
24  80  36  75   2  69  90  31  96   2
61  93  82  45  69  31  83  84  53  90
49  74   1  94  38  55  65  57  89  20
90  18  14  47  84  16  82  38  18  30
77  41  21  42  51  70  67  71  98  67
46  94  20  85  71  38  35  55  28  29
 2  92  61  53  43  87  29  45  26  47
83  42  28  21  31  86  35  70  88   7
45  90  20  68  19  60  54  63  74  99
```

Note: an input file contains only one sequence but is formatted into different lines of aligned numbers for better reading convenience.

Corresponding output:

```
18
```