Gathering the artifacts

Let we have a square labyrinth of size n×n (n≤20) without exits to outside. This labyrinth has n^2 chambers enumerated from 1 to n^2 in order from left to right and from above to below. Between two neighboring (by vertical or by horizontal) chambers may be:

- a wall;
- a passage;
- a door, initially locked.

The keys of the doors are scattered in the chambers. A chamber may contain several keys. There are keys of k different types (k≤9). The key of the type i (1≤i≤k) can open any door with the lock of this type (from any side). After unlocking, you can't extract the key from the keyhole, but the door stay opened. The number of keys of any type is equal to the number of the doors with the lock of this type.

Further, m (m≤50) artifacts are scattered in the chambers (in a chamber may be any number of artifacts).

At the beginning, a player is in the left top chamber of the labyrinth. He may:

- pass from any chamber to any other neighboring trough the passages or opened doors;
- pick up keys and artifacts in the chamber where he is;
- open a door if he has the appropriate key.

The game stops when the player picks up the last artifact.

We call a moving of the player between two neighboring chambers *a step*. The player's goal is to gather the artifacts with the minimal number of steps.

Input text file ARTIFACT.IN consists of a description of one test. This description includes:

- one line with three numbers separated by spaces: size of labyrinth (n), number of key types (k), and number of artifacts (m);
- **n** lines with the labyrinth structure. Each line describes the chambers of one labyrinth horizontal. It contains two characters for one chamber specifying the eastern and southern walls of this chamber by next codes:

- W – solid wall,
- P – passage,
- digit i (i≤k, i≠ 0) – door with the lock of type i.

  The descriptions of the chambers in the line (separated by commas) are going from left to right. The order of these lines corresponds to an order of horizontales from above to below;

- k lines, one for each key type, with the chamber numbers (separated by spaces) where the keys of this type are scattered;
- one line with the chamber numbers (separated by spaces) where the artifacts are scattered.

Output text file ARTIFACT.OUT must contain the minimal number of steps to gather the all artifacts, or -1 if it's impossible to do that.

An example of input and output data.
Let the labyrinth have a structure demonstrated in a figure, where symbols **a** indicate positions of scattered artifacts. Input file will be:

```
4 2 3
WP,WP,PP,WP
```

```
PW,WP,WW,WP
WW,WP,WW,WP
2W,PW,1W,WW
2
16
8 10 16
```

Output file for this example must contain a line with number 10

Last digit

For a number of the form $3^k \cdot N!$, we are given N and the last nonzero digit Y. Moreover, we are given a number M. Numbers N, Y, M are natural, $1 \le N \le 1000000000$, $1 \le M \le 10000$, $1 \le Y \le 9$.

Your program must determine the last nonzero digit of number $3^k \cdot (N+M)!$

Input text file DIGIT.IN contains a single line with values of N, Y, M, separated by spaces.

Output text file DIGIT.OUT contains the last nonzero digit.

Input sample	Output sample
3 6 1	4

Inverse Matrix

Let A be nxn nonsingular matrix. We are given the inverse matrix A^{-1} (A is unknown). Suppose that a matrix B is obtained from A by substituting a row A_i for a row vector a.

Your program must determine whether B is nonsingular; if the answer is affirmative, compute B^{-1} to within 10^{-6}.

Input text file INVERSE.IN contains:

- // An ordered list of triples representing the nonzero entries of $A^{-1} = [v_{i,j}]$:

n // size of the matrix
k // number of the entries in the list
i1 *j1* *v1* // the first entry, $v1 = v_{i1,j1}$
i2 *j2* *v2* // the second entry, $v2 = v_{i2,j2}$
…
i_k j_k *vk* // the last entry, $vk = v_{ik,jk}$
// The pairs (*i1*, *j1*), (*i2*, *j2*), …,(*ik*, *jk*) are lexicographically ordered,
// i.e. if (*i,j*) and (*k,l*) are two successive pairs in the list,
// then *i* ? *k,* and if *i*=*k,* then *j* < *l*.

- // A row index ***i*** and a row vector ***a***:

i // index of a row
a1 a2 … an // *n* numbers

// The all numbers in the input file are integers separated by spaces and/or line breaks.

Output text file INVERSE.OUT contains:

- a line with the Yes/No answer;

- a list of the nonzero entries of B^1 in the same form as the input list for A^{-1}. The all entries of B^1 with absolute values less than 10^{-6} must be skipped.

Example.

File INVERSE.IN:

```
4
1  1  1   1  2  -2   1  4
2    2    –1    2    3
3  1  4   3  3  2   3  4
4    1    1    4    4
3
0 1 –1 2
```

File INVERSE.OUT:

```
10 Yes
1 4                    13
3 1  1  1   1  2  –2   1  4  1
1 2 1 6   2 2 2   2 3 –3 2 4 -3
1 3 1 8   3 2 2   3 3 –2  3 4 -1
4 1 1  4 4 1
```

Intersection of lines

We are given n straight lines described by their end points (2≤n≤20) in the plane. Your program must determine a number of different points where lines are intersected. The accuracy of calculations is to be within of 10^{-4}.

The first line of the input text file LINES.IN contains n. Next n lines contain the coordinates of the endpoints of n straight lines in the form

x1 y1 x2 y2

where x1, y1 are coordinates of the first endpoint, x2, y2 are coordinates of the second one. These real numbers can have up to 5 digits after decimal points. They are separated by spaces.

Output text file LINES.OUT contains the number of different points where lines are intersected; if this number is infinite, then the file must contain a single line with number -1.

Input sample

```
2
0.0          0.0          2.0          2.0
1.0 0.0 0.0 1.0
```

Output sample

```
1
```

Input sample

```
2
0.0          0.0          2.0          2.0
1.0 1.0 2.0 2.0
```

Output sample

```
-1
```

String compression

Given a string X of symbols of Latin alphabet; its length is not greater than 255. X can be compressed by the following algorithm. If the string contains N consecutive equal substrings S of K symbols (1≤K,N≤9), we can substitute them by digit K, digit N and substring S.

This operation can be applied many times (but not for substrings compressed before) in order to get a string with the minimal possible length.

For example, string abababababccccccc can be compressed into 25ab16c, or 42abababab23cc, or a42babab32ccc.

Your program must determine a compressed string with minimal possible length. In case of many solutions, you must give the first of them in lexicographical order.

Input text file SQUEEZE.IN contains a string X.

Output text file SQUEEZE.OUT contains the compressed string.

Input sample	Output sample
aaaaaaaaaa	19aa
abcde	abcde

Theatre tickets

One theatre provides n ($3 \le n \le 50$) different performances. You need exactly a_i tickets for the performances of type i (i =1, …, n, $0 \le a_i \le 400$, integer). But you can get some free tickets by the following way. If you buy at once k tickets for the different performances, you can get one free ticket for any performance you want.

Your program must determine the maximal possible number of tickets you can get free.

The first line of input text file TICKETS.IN contains n and k. The second line contains values a_i (i =1, …, n). All values are separated by space .

Output text file TICKETS.OUT contains the maximal number of free tickets.

Input sample	Output sample
3 3 2 2	3 1