task	osmo	waclaw	lampice	sibice

source file	`osmo.pas`
	`osmo.c`
	`osmo.cpp`
		`waclaw.pas`
		`waclaw.c`
		`waclaw.cpp`
			`lampice.pas`
			`lampice.c`
			`lampice.cpp`
				`sibice.pas`
				`sibice.c`
				`sibice.cpp`

input data	stdin			
output data	stdout			
time limit (Athlon MP 2x2.1 GHz)	1 sec			
memory limit (heap)	32 MB			
memory limit (stack)	8 MB			
points	**30**	**45**	**55**	**70**
	200			

Word search is a very popular enigmatic game that consists of a table with letters arranged in N rows and N columns and a list of words we have to search for.

To solve an instance of word search we, for each word from the list, find **all** appearances of that word in the table and cross out **all** the letters forming that particular appearance. The word can start anywhere and lay in any of the **eight directions** (up, down, left, right and the four diagonal directions).

Reading **all non-crossed letters** in row-major order (up to down and in each row from left to right) gives the solution of the word search puzzle.

Write a program that will find the solution of the given puzzle.

input data

The first line of input contains an integer N, $1 \leq N \leq 10$, the size of the table.

Each of the next N lines contains a sequence of N characters representing one row of the table.

The next line contains an integer R, $1 \leq R \leq 100$, the number of words in the list.

Each of the next R lines contains one word from the list, at most 10 characters long.

All of the letters in the word search and all the words in the list will consist of lowercase letters of the English alphabet ('a'-'z').

output data

The first and only line of output should contain the solution of the puzzle.

Note: the test data will be such that a solution of at least one letter will always exists.

examples

input

```
2
ab
cd
1
ad
```

output

```
bc
```

input

```
5
patka
guska
macka
klopa
krava
4
patka
guska
macka
krava
```

output

```
klopa
```

input

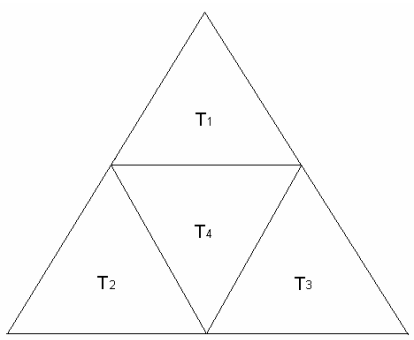
```
4
rast
plso
tajv
pnez
6
rast
alan
nt
past
taj
bonus
```

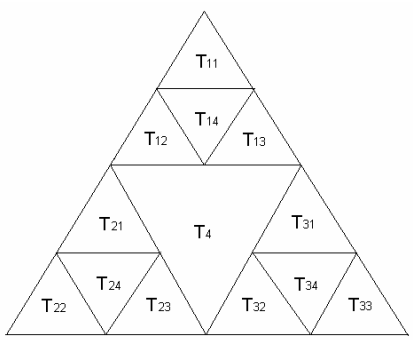
output

```
povez
```

Waclaw Sierpinski was a Polish mathematician who liked playing with triangles. One day he started drawing triangles using the following procedure:

- Draw an equilateral triangle T.
- Connect the midpoints of its sides with line segments. Denote the new equilateral triangles with T1, T2, T3 and T4, as illustrated in the first figure below.
- Repeat the previous step on triangles T1, T2 and T3. New triangles are: T11, T12, T13, T14, T21, T22, T23, T24, T31, T32, T33, T34.
- Continue the procedure on all triangles ending in 1, 2 or 3. The resulting fractal is called the Sierpinski triangle.





We say that a triangle A is **leaning** on the triangle B if B **does not contain** A and if one **entire** side of A **is a part** of some side of B. For example, the triangle T23 is leaning on T24 and T4, but not on T2 or T32. Note that A leaning on B does not imply that B is leaning on A.

Write a program that, given a triangle A, which is a part of the Sierpinski triangle, finds **all** triangles B such that A is leaning on B.

input data

The first and only line of input contains a sequence of characters representing the given triangle, as described above. The sequence will contain between 2 and 50 characters, inclusive.

output data

Output all triangles that the given triangle is leaning on, each on a separate line, in any order.

examples

input	**input**	**input**
T4	T11	T312
output	**output**	**output**
T1 T2 T3	T14	T4 T314 T34

2*N light bulbs are arranged in two rows and N columns. Each light bulb can be either off or on, and all lights are initially off.

We want to turn some of them on so that they form a beautiful pattern. In one step we can **change the state** of a sequence of (one or more) **consecutive** light bulbs in the same row or column.

Given the desired pattern, write a program that finds the **minimum** number of steps required to form the pattern.

The following figure illustrates the seven steps needed to obtain the pattern given in the third example:

0	**1**	**2**	**3**
00000000000000000000 00000000000000000000	**111**00000000000000000 00000000000000000000	11100010000000000000 00000010000000000000	11100010000000000000 0**1111101**100000000000
4	**5**	**6**	**7**
11101101111000000000 01111101100000000000	11101101111000**1111**0 01111101100000000000	11101101111000**1**01110 01111101100000**1**0000	11101101111000101**0**10 01111101100000010**1**00

input data

The first line of input contains an integer N, $1 \leq N \leq 10{,}000$, the number of columns.

Each of the following two lines contains a sequence of N characters representing the desired final pattern.

Character '1' indicates a light bulb that should be on in the final state, while the character '0' indicates a light bulb that should be off.

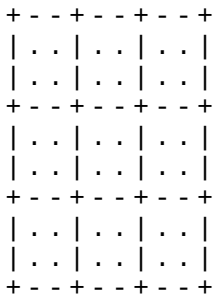
output data

The first and only line of output should contain a single integer – the minimum number of steps required.

examples

input	**input**	**input**
3 100 000	5 11011 11011	20 11101101111000101010 01111101100000010100
output	**output**	**output**
1	3	7

24 matches are arranged to form a 3x3 grid as illustrated in the figure below.



Two consecutive '-' (minus) characters represent a horizontal match, whereas two consecutive '|' (vertical bar) characters represent a vertical match. '+' (plus) characters represent positions where two or more matches' ends touch. The interior of the board is filled with '.' (dot) characters.

Write a program that, given integers N and K, **removes exactly N matches** in such a way that the remaining matches form **exactly K squares** and that **each remaining match is part of some square**.

input data

The first and only line of input contains two integers N and K, $1 \leq N < 24$, $1 \leq K < 14$, the number of matches to be removed and the number of squares wanted.

output data

Output the resulting board in a format identical to the figure from the task description, with dot characters in place of the removed matches.

Note: the test data will be such that a solution, although not necessarily unique, always exists.

examples

input

```
20 1
```

output

```
+--+..+..+
|..|......
|..|......
+--+..+..+
..........
..........
+..+..+..+
..........
..........
+..+..+..+
```

input

```
5 4
```

output

```
+--+--+--+
|..|..|..|
|..|..|..|
+--+--+..+
|.....|..|
|.....|..|
+--+--+..+
|........|
|........|
+--+--+--+
```

input

```
4 6
```

output

```
+--+--+--+
|..|..|..|
|..|..|..|
+--+--+..+
|..|..|..|
|..|..|..|
+--+--+..+
|........|
|........|
+--+--+--+
```