Problem J: Consecutive ones

```
00000000000000000011
11111111000000000000
00000000000000001111
00000000000011000000
00000000000000111100
00000000000001110000
00111000000000000000
00000000000111000000
00000000111100000000
00000000000000000001
11000000000000000000
00001111111000000000
00000111111111111111
00000000011111100000
00000000001111111110
00000000000000011110
00000001111100000000
00000011111111110000
00011110000000000000
01111111111100000000
00000000000000000111
```

A time schedule is represented by a 0-1 matrix with n lines and m columns. Each line represents a person and each column an event. All the persons participating to an event have a one in the corresponding entry of their line. Persons not attending the event have a zero entry in that column. Events occur consecutively.

Problem

Write a program that finds a *smart permutation* of the events where each person attends all its events in a row. In other words, permute the columns of the matrix so that all ones are consecutive in each line.

Input specification

The first line of the input consists in the number $n \leq 400$ of lines. The second line contains $m \leq 400$, the number of columns. Then comes the n lines of the matrix. Each line consists in m characters 0 or 1.

The input matrix is chosen so that there exists only one smart permutation which preserves column 0 in position 0. To make things easier, any two columns share few common one entries.

Output specification

The output consists of m numbers indicating the smart permutation of the columns. The first number must be 0 as column 0 does not move. The second number indicate the index (in the input matrix) of the second column, and so on.

Sample input

```
3
4
0110
0001
1101
```

Sample output

```
0
3
1
2
```

Sample input

```
6
5
01010
01000
10101
10100
00011
00101
```

Sample output

```
0
2
4
3
1
```

Sample input

```
21
20
00101000000000000000
10010010010110010100
00101110100000000000
01000000000000001000
00000101100000100000
01000000100000100000
00000010000110000000
01000000000001001000
00000000001001000011
00001000000000000000
10000000000000000100
00010010011000010011
01111101111001111011
01000000000001101011
01100101100001101001
00100101100000000000
00010000001001000011
01010000101001111011
00000010010010010000
00010010011111010111
00101001000000000000
```

Sample output

```
0
17
11
12
6
9
15
3
10
18
19
13
16
1
14
8
5
7
2
4
```