| task | zbroj | vlak | tetris | krug |
|---|---|---|---|---|
| **source code** | `zbroj.pas`<br>`zbroj.c`<br>`zbroj.cpp` | `vlak.pas`<br>`vlak.c`<br>`vlak.cpp` | `tetris.pas`<br>`tetris.c`<br>`tetris.cpp` | `krug.pas`<br>`krug.c`<br>`krug.cpp` |
| **input** | standard input | | | |
| **output** | standard output | | | |
| **time limit** | 1 second | | | |
| **memory limit** | 32 MB (heap) + 8 MB (stack) | | | |
| **points** | **30** | **40** | **60** | **70** |
| | **200** | | | |

# ZBROJ

Young Borko's mother is making him work on his arithmetic. In spite of the large amount of time he spends pretending to be practicing, he still can't quite manage addition.

Borko is aware that he has to add one digit at a time from right to left, but he's too lazy to "carry over" so he always immediately adds the sum of two digits to the left of the result.

For example, he adds the numbers 123 and 2495 like this:

```
    1  2  3
+ 2  4  9  5
-------------
  2  5 11  8
```

When he reads the result, he gets 25118.

You are given two numbers which Borko is to add. Write a program that determines the result he will get.

## Input

The first and only line of input contains two integers A and B separated by a space, the numbers Borko is trying to add. Both numbers will be between 1 and 1 000 000.

## Output

Output the "sum" Borko gets.

## Example test cases

| **input** | **input** | **input** |
|---|---|---|
| 512 444 | 123 2495 | 99999 99999 |
| **output** | **output** | **output** |
| 956 | 25118 | 1818181818 |

# VLAK

An empty train on a train station is waiting for passengers to board. The train consists of N cars and each car can accommodate K passengers.

The train station is swamped with strange demands. Passengers board the train one by one and go to one of the cars in which there are free seats left. Each passenger chooses a car to board according to the following rules:

1. Of all the cars which are not full, the passenger chooses the one which has the fewest passengers **whose names start with the same letter** as his.
2. If there are multiple cars which have the same fewest number of such passengers, the passenger chooses the one with the **smallest total number** of passengers.
3. If there are still multiple cars, the passenger boards the first of these.

Write a program that, given the list of passengers which are boarding the train, determines **the number of passengers in each car** after boarding has completed.

## Input

The first line contains two integers N and K, $1 \leq N \leq 10$, $1 \leq K \leq 10$, the number of cars and the number of passengers each car can accomodate.

The second line contains an integer P, $1 \leq P \leq N \cdot K$, the number of passengers waiting to board.

The next P lines contain the names of all passengers, in order in which they will board. The name of each passenger is a string of at most 10 lowercase letters of the English alphabet. No two passengers will have the same name.

## Output

Output N integers separated by single spaces on a single line, the number of passengers in each car, in order from first car to last.

## Example test cases

**input**

```
3 2
3
anka
branko
cvjetko
```

**output**

```
1 1 1
```

**input**

```
2 2
4
mirko
nenad
ozren
nino
```

**output**

```
2 2
```

**input**

```
4 3
6
goranka
mirko
marin
antun
petar
mladen
```

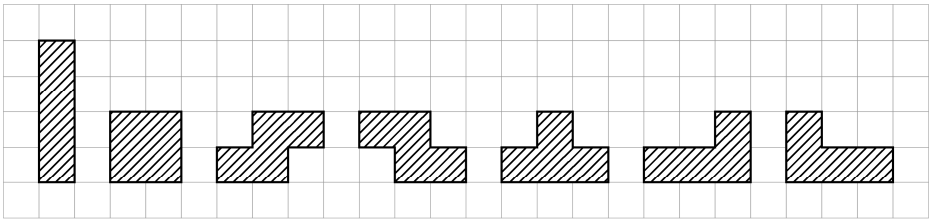
**output**

```
2 1 1 2
```

**Clarification for first example**: When Anka is boarding, the train is empty; according to rules 1 and 2 she can choose any car, so she chooses the first car according to rule 3. Rule 1 also allows Branko to entery any car, but rule 2 limits him to the second and third. By rule 3 he enters the second car. Cvjetko enters the third car by rule 3, since it is empty.

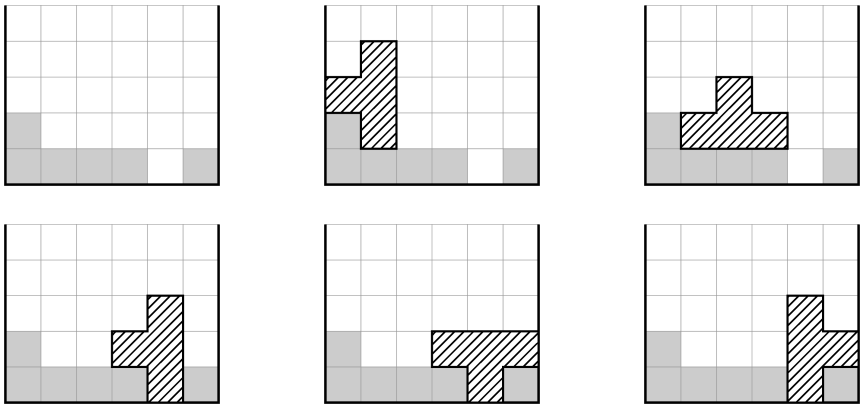
# TETRIS

Tetris is a popular computer game played in a field consisting of C columns and an unlimited number of rows. In one move, one of the following seven pieces is dropped into the field:




When dropping the piece, the player is free to rotate the piece 90, 180 or 270 degrees and to move it left or right, as long as the piece stays entirely in the field. The piece then falls until it settles on the bottom of the field or on already occupied squares. In our variant of Tetris the piece **must** fall so that **all parts** of the piece are on the bottom of the field or on already occupied squares. In other words, after the piece falls there may not be **a free square** such that **some square above it is occupied**.

For example, let the field be six columns wide with initial heights (the number of already occupied squares in each column) 2, 1, 1, 1, 0 and 1. Piece number 5 can then be dropped into the field in five different ways:



You are given the initial heights of all columns and the figure to be dropped into the field.

Write a program that calculates the number of different ways to do this, i.e. the number of different field configurations that can be achieved by dropping the piece.

## Input

The first line contains two integers C and P, $1 \leq C \leq 100$, $1 \leq P \leq 7$, the number of columns and the number of the piece to be dropped.

The second line contains C integers separated by single spaces, each between 0 and 100, inclusive. These are the initial heights of the columns.

## Output

Output on a single line the number of different ways to drop the piece in the field.

## Example test cases

**input**

```
6 5
2 1 1 1 0 1
```

**output**

```
5
```

**input**

```
5 1
0 0 0 0 0
```

**output**

```
7
```

**input**

```
9 4
4 3 5 4 6 5 7 6 6
```

**output**

```
1
```

One nice summer day while Mirko was drinking lemonade in his room...

"Big brother!", yells Stanko.

"I wonder sometimes which of the two of us is the big one. What is it?", Mirko asked.

"Listen carefully! In the backyard I have N pebbles arranged in a circle. Some of the pebbles are black, some are white. I will do the following: between any two neighbouring pebbles of the **same colour** I will insert a **black pebble**, and between any two neighbouring pebbles of **different colours** I will insert a **white pebble**. At that point there will be 2N pebbles in the circle, so I will remove the starting N pebbles so that only the newly added N pebbles remain. And all this I intend to do **exactly K times**. And then you are to determine my starting circle.", said Stanko long-windedly.

"Ha! I shall not fall prey to your trickery! I can see that it is not necessarily possible to know exactly what the starting circle was, but I can count the number of distinct starting circles that give the same result as your circle after exactly K of those weird transformations of yours", answered Mirko.

You are given the configuration of the circle **before** Stanko performed the transformation described above K times.

Write a program that determines the number of distinct starting circles that give the same circle after K transformations as Stanko's original circle does after K transformations.

Two configurations of pebbles are **considered to be the same circle** if one can be gotten from the other by rotating it any number of positions. For example BBW and BWB are the same circle whereas BBWWBW and WWBBWB are not.

## Input

The first line of input contains two integers N and K, $3 \leq N \leq 100$, $1 \leq K \leq 10$, where N is the number of pebbles in the circle and K is the number of transformations made by Stanko.

The second line contains exactly N characters 'B' or 'W' representing Stanko's original circle.

## Output

Output the number of possible distinct starting circles on a single line.

## Example test cases

| **input** | **input** |
|---|---|
| 3 1<br>CCB | 6 2<br>BCBBCB |
| **output** | **output** |
| 2 | 3 |

**Test case 1 clarification**: The circles BBW and WBW become circle BWW after one transformation.