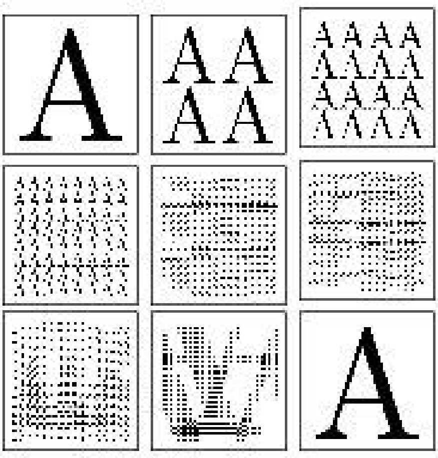
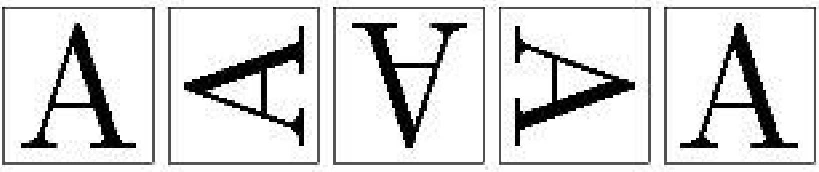
Shuffling the pixels in a bitmap image sometimes yields random looking images. However, by repeating the shuffling enough times, one finally recovers the original images. This should be no surprise, since "shuffling" means applying a one-to-one mapping (or permutation) over the cells of the image, which come in finite number.



Your program should read a number $n$, and a series of elementary transformations that define a "shuffling" $\phi$ of $n \times n$ images. Then, your program should compute the minimal number $m$ ($m > 0$), such that $m$ applications of $\phi$ always yield the original $n \times n$ image.

For instance if $\phi$ is counter-clockwise $90^o$ rotation then $m = 4$.

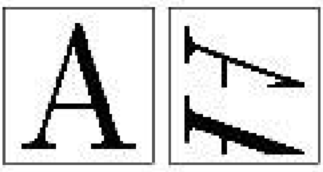


## Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

Input is made of two lines, the first line is number $n$ ($2 \le n \le 2^{10}$, $n$ even). The number $n$ is the size of images, one image is represented internally by a $n \times n$ pixel matrix ($a_i^j$), where $i$ is the row number and $j$ is the column number. The pixel at the upper left corner is at row 0 and column 0.

The second line is a non-empty list of at most 32 words, separated by spaces. Valid words are the keywords **id**, **rot**, **sym**, **bhsym**, **bvsym**, **div** and **mix**, or a keyword followed by "**-**". Each keyword key designates an elementary transform (as defined by Figure 1), and **key-** designates the inverse of transform **key**. For instance, **rot-** is the inverse of counter-clockwise $90^o$ rotation, that is clockwise $90^o$ rotation. Finally, the list $k_1, k_2, \ldots, k_p$ designates the compound transform $\phi = k_1 \circ k_2 \circ \cdots \circ k_p$. For instance, "**bvsym rot-**" is the transform that first performs clockwise $90^o$ rotation and then vertical symmetry on the lower half of the image.



**id** , identity. Nothing changes : $b_i^j = a_i^j$.

**rot** , counter-clockwise $90^o$ rotation

**sym** , horizontal symmetry : $b_i^j = a_i^{n-1-j}$

**bhsym** , horizontal symmetry applied to the lower half of image : when $i \ge n/2$, then $b_i^j = a_i^{n-1-j}$. Otherwise $b_i^j = a_i^j$.

**bvsym** , vertical symmetry applied to the lower half of image ($i \ge n/2$)

**div** , division. Rows $0, 2, \ldots, n-2$ become rows $0, 1, \ldots n/2-1$, while rows $1, 3, \ldots n-1$ become rows $n/2, n/2+1, \ldots n-1$.

**mix** , row mix. Rows $2k$ and $2k+1$ are interleaved. The pixels of row $2k$ in the new image are $a_{2k}^0, a_{2k+1}^0, a_{2k}^1, a_{2k+1}^1, \cdots a_{2k}^{n/2-1}, a_{2k+1}^{n/2-1}$, while the pixels of row $2k+1$ in the new image are $a_{2k}^{n/2}, a_{2k+1}^{n/2}, a_{2k}^{n/2+1}, a_{2k+1}^{n/2+1}, \cdots, a_{2k}^{n-1}, a_{2k+1}^{n-1}$.

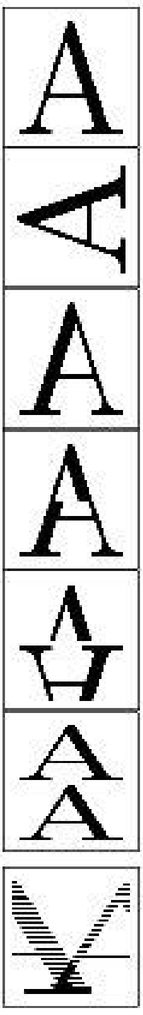


Figure 1: Transformations of image ($a_i^j$) into image ($b_i^j$)

## Output

For each test case, your program should output a single line whose contents is the minimal number $m$ ($m > 0$) such that $\phi^m$ is the identity. You may assume that, for all test input, you have $m < 2^{31}$.

The outputs of two consecutive cases will be separated by a blank line.

## Sample Input

```
2

256
rot- div rot div

256
bvsym div mix
```

## Sample Output

```
8

63457
```