code warrior

Infrastructure CodeWarrior Challenge #2

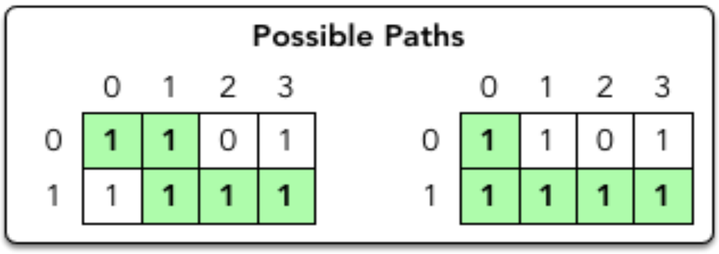
Problem #1: Tell me the path
Score points: 100

A forklift worker moves products from one place to the other in an automotive parts warehouse. There a map in the dashboard that shows, in real time, the open and blocked sections inside the warehouse. The map is displayed as an n x m matrix of 1's and 0's which represent open and blocked sections respectively. A forklift driver always starts at the upper left corner of the map at warehouse[0][0] and tries to reach the bottom right section of the map or warehouse[n-1][m-1]. Each movement must be in increasing value along a row or column but not both. Given the warehouse map, determine the number of distinct paths to get from warehouse[0][0] to warehouse[n-1][m-1]. The number may be large, so return the value modulo (109+7).

Example

Consider the matrix below, showing open and blocked sections of the warehouse: 1 indicates an open section and 0 indicates a blocked section.

- It is only possible to travel through open sections, so no path can go through the section at (0, 2).
- There are two distinct paths to the goal from position(0,0) to the position (1,3). 2 modulo (109+7) = 2



There are two possible paths from warehouse[0][0] to warehouse[1][3].

Function Description

Complete the function numPaths in the editor below.

numPaths has the following parameter(s):

warehouse[n][m]: a two dimensional array of integers of n rows and m columns

Returns:

int: the number of paths through the matrix, modulo (109 + 7).

Constraints

- 1 ≤ n, m ≤ 1000
- Each cell in matrix a contains either a 0 or a 1.

Sample Case 0

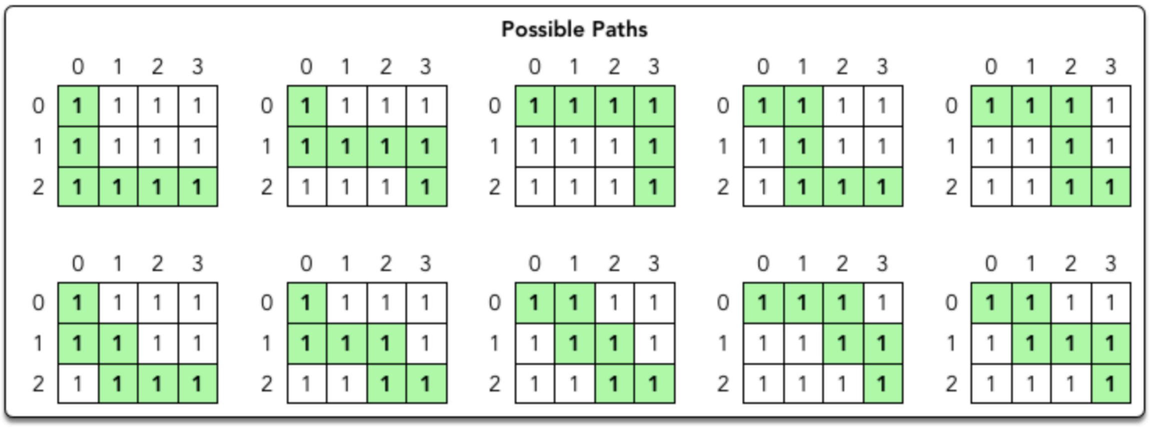
Sample Input 0

```
STDIN       Function
-----       ---------
3      →    warehouse[][] size n=3 m=4
4
1 1 1 1 →   warehouse = [[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]]
1 1 1 1
1 1 1 1
```

Sample Output 0

10

Explanation 0



There are *10* possible paths from *warehouse[0][0]* to *warehouse[2][3]*.

10 modulo (10^9+7) = 10

Problem #2: Magic Script
Score points: 100

If you've ever looked at an assembly language dump of an executable file, you know that commands come in the form of hexadecimal digits that are grouped by the processor into instructions. It is important that the parsing be done correctly or code will not be executed as expected. Wrong parsing is the basis for Return Oriented Programming, a method of causing mahem.

You have developed a program in a new scripting language. Of course, it requires accurate parsing in order to perform as expected, and it is very cryptic. You want to determine how many valid commands can be made out of your lines of script. To do this, you count all of the substrings that make up a valid command. Each of the valid commands will be in the following format:

- The first letter is a lowercase English letter.
- Next, it contains a sequence of *zero or more* of the following characters: lowercase English letters, digits, and colons.
- Next, it contains a forward slash '/'.
- Next, it contains a sequence of *one or more* of the following characters: lowercase English letters and digits.
- Next, it contains a backward slash '\'.
- Next, it contains a sequence of *one or more* lowercase English letters.

Given some string, *s*, we define the following:

- *s[i..j]* is a substring consisting of all the characters in the inclusive range between index *i* and index *j*.
- Two substrings, *s[i[1]..j[1]]* and *s[i[2]..j[2]]*, are said to be *distinct* if either *i[1] ≠ i[2]* or *j[1] ≠ j[2]*.

Example 0
commands = ['*abc:/b1c\xy', 'w:/a\bc/23a\de::/12\xyz'*]

The 6 valid commands in the *commands[0]* string are:

```
commands[0] = abc:/b1c\xy
abc:/b1c\xy
bc:/b1c\xy
c:/b1c\xy
abc:/b1c\x
bc:/b1c\x
c:/b1c\x
```

The 10 valid commands in the *commands[1]* are:

```
commands[1] = w:/a\bc/23a\d:e:/12\xyz
w:/a\bc
w:/a\b
bc/23a\d
c/23a\d
d:e:/12\xyz
d:e:/12\xy
d:e:/12\x
e:/12\xyz
e:/12\xy
e:/12\x

Note that the first command overlaps the second at bc.  The second overlaps the third at d.
```

The first string can be parsed 6 ways, and the second can be parsed 10 ways. The return array is [6, 10].

Function Description

Complete the function *commandCount* in the editor below. The function must return an array of integers, each representing the number of valid command strings in *commands[i]*.

commandCount has the following parameter(s):

string commands[n]: an array of strings

Constraints

- $1 \le n \le 50$
- Each character of *commands[i] is in the set {a-z,0-9,/,\,:}.*
- The length of each *commands[i]* $\le 5 \times 10^5$.

Input Format

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the array *commands*.

Each of the next *n* lines contains a string *commands[i]*.

Sample Case 0

Sample Input 0

```
STDIN          Function
-----          --------
6          →   commands[] size n = 6
w\\//a/b   →   commands[] = ['w\\//a/b', 'w\\//a\b', 'w\\/a\b', 'w::/\a\b', 'w::/a\b', 'w:/a\bc::/12\xyz']
w\\//a\b
w\\/a\b
w::/\a\b
w::/a\b
w:/a\bc::/12\xyz
```

Sample Output 0

```
0
0
0
0
1
8
```