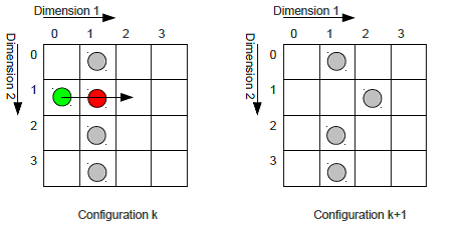
N-dimensional board game

Consider a *Peg Solitaire* like board game, where a token is moved from a field to an empty field by passing a neighbor token, that is in turn removed. A move thus requires an occupied field next to a token and an unoccupied field next to its neighbor in the same direction.

The board has a range of *D* (2<*D*<10) fields in *N* (0<*N*<10) dimensions (thus having *DN* fields in total). Each field may be occupied by a token, whose position is denoted with descending dimension: [*p(N) p(N-1) ... p(2) p(1)*]

The example below illustrates a 2-dimensional board of size *D*=4 and the transition from a configuration *k* to another configuration *k+1*.



In the example, the token on position [1 0] (green) is moved to position [1 2] and token [1 1] (red) is removed. Obviously, the number of tokens is decremented with every move.

A token can be moved upwards and downwards in any dimension, the maximum number of possible moves of a token is thus 2 D.

Every possible move of a configuration branches to a new configuration.

The game is considered to be *lost* if no moves are possible from a configuration and more than one token is left, or to be *won* if a single token is resides in the configuration.

Task

Write a program that reads a start configuration (a single line from *stdin*) and outputs the the **number of possible won games** (i.e. configurations with only one token left) to *stdout* (0 if no solution is found).

Input / Output format

A start configuration is denoted by a string, starting with size and dimensionality  (separated by '-', followed by ':'). Then the occupation of all fields of the configuration is given, while each dimension >1 is enclosed by braces '{'/'}' as shown in the table below. An occupied field is indicated by '1', an empty field by '0', separated by space characters. The input is terminated by '$'.

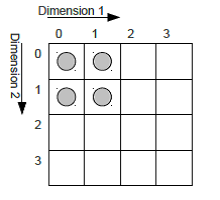
|  |  |
| --- | --- |
| 1-D | “D-1:x x x$” |
| 2-D | “D-2:{x x x}{x x x}{x x x}$” |
| 3-D | “D-3:{{x x x}{x x x}{x x x}}{{x x x}{x x x}{x x x}}{{x x x}{x x x}{x x x}}$” |

**Example input**, compare to sketch in the bottom:

4-2:{0 0 1 1}{0 0 1 1}{0 0 0 0}{0 0 0 0}$

**Expected output:**

integer number as string.



Example 1

**Input:**

4-1:1 0 1 1

**Output:**

1

One possible solution with

two moves as shown on

the right.

