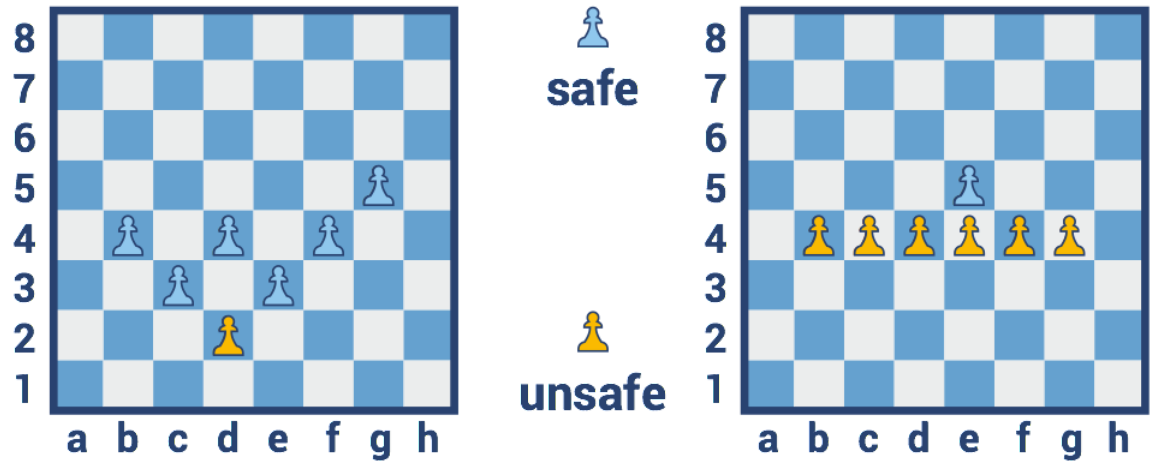
Almost everyone in the world knows about the ancient game Chess and has at least a basic understanding of its rules. It has various units with a wide range of movement patterns allowing for a huge number of possible different game positions (for example Number of possible chess games at the end of the n-th plies.) For this mission, we will examine the movements and behavior of chess pawns.

Chess is a two-player strategy game played on a checkered game board laid out in eight rows (called ranks and denoted with numbers 1 to 8) and eight columns (called files and denoted with letters a to h) of squares. Each square of the chessboard is identified by a unique coordinate pair — a letter and a number (ex, "a1", "h8", "d6"). For this mission we only need to concern ourselves with pawns. A pawn may capture an opponent's piece on a square diagonally in front of it on an adjacent file, by moving to that square. For white pawns the front squares are squares with greater row number than the square they currently occupy.

A pawn is generally a weak unit, but we have 8 of them which we can use to build a pawn defense wall. With this strategy, one pawn defends the others. A pawn is safe if another pawn can capture a unit on that square. We have several white pawns on the chess board and only white pawns. You should design your code to find how many pawns are safe.



You are given a set of square coordinates where we have placed white pawns. You should count how many pawns are safe.

Input: Placed pawns coordinates as a set of strings.

Output: The number of safe pawns as a integer.

Example:

```
safe_pawns({"b4", "d4", "f4", "c3", "e3", "g5", "d2"}) == 6
safe_pawns({"b4", "c4", "d4", "e4", "f4", "g4", "e5"}) == 1
```

How it is used: For a game AI one of the important tasks is the ability to estimate game state. This concept will show how you can do this on the simple chess figures positions.

Precondition:
$0 < \text{pawns} \le 8$