Crossword Puzzle

Catherine Javadian, Jennifer Cafiero,
And Jordana Approvato

The Problem (Formal)

A Crossword grid is provided to you, along with a set of words (or names of places) which need to be filled into the grid. The cells in the grid are initially, either `+` signs or `-` signs. Cells marked with a `+` have to be left as they are. Cells marked with a `-` need to be filled up with an appropriate character.

The Problem (Simplified)

Solve a crossword puzzle!

Input

The input contains lines, each with characters (which will be either + or - signs).

After this follows a set of words (typically nouns and names of places), separated by semi-colons (;).

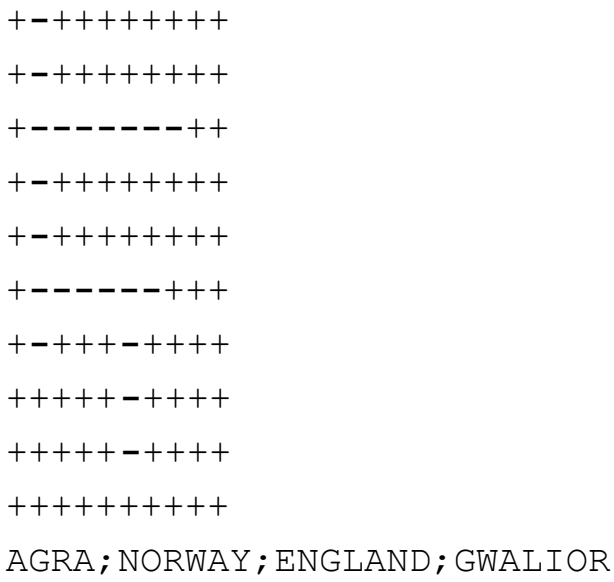
Constraints

There will be no more than ten words. Words will only be composed of upper-case A-Z characters. There will be no punctuation (hyphen, dot, etc.) in the words.

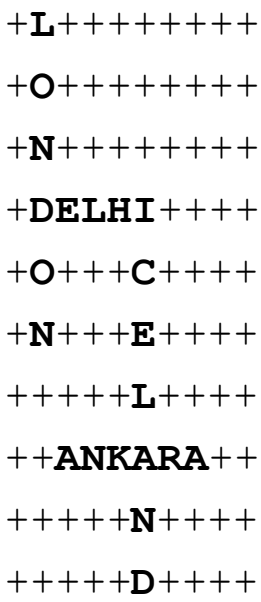
Output

Position the words appropriately in the grid, and then display the grid as the output. So, your output will consist of 10 lines with 10 characters each.

Test Cases



Solutions



The Language

Struggles/Realizations

Struggles:

- Dealing with vertical and horizontal words (Added a vertical boolean

Working:

- Modified form of scramble squares

Algorithm Explanation

- Uses recursion to go through all possible combinations
- Checks if word can fit into the current space, if it can than it adds that word to the spaces and loops back through, working off of the grid with the word added
- Returns false from possible combination if no word will fit into a space/if puzzle is filled but there are still words left
- Returns true if part of combination will fit and continues to recurse through that possible combination with the remaining words

Code Walkthrough (The Good Stuff)

```
def findSol(grid, spaces, words):
    '''Finds the solution of the puzzle. If words is empty (len 0), returns
       True to indicate that the puzzle is solved.'''
    if len(words) == 0:
        return True
    for word in words:
        for space in spaces:
            if canAddWord(grid, space, word):
                before = addWord(grid, space, word)
                space.taken = True
                new_words = set(words)
                new_words.remove(word)
                if findSol(grid, spaces, new_words):
                    return True
                addWord(grid, space, before) #will revert the grid if the solution is incorrect
                space.taken = False

    #if it reaches this far, breaks out of the for loop, and does not yet have a solution, it is unsolvable
    return False
```

Demo