# Word Search v2

Your task is to produce a solution to a word search problem; this is the sort of thing printed in crossword magazines and so on. You're given a square grid of letters and a search word and the aim is to be able to work out if that word appears horizontally or vertically in the grid.

The grid consists only of the 26 lowercase letters in the range `a...z` and the 'words' are not necessarily dictionary words, just random strings of between 4 and 20 characters.

There are no tricks here, we're only interested in words which appear vertically from the top down in a single column or horizontally left to right in a single row.

We suggest you start with the following Python implementation and please keep the core interface the same:

```
class WordSearch:
    def __init__(self, grid: str):
        pass
    def is_present(self, word: str) -> bool:
        return True
```

In the `__init__()` method, the `grid` parameter is a string value representing the entire grid content, starting from the top row, with each row concatenated. There are no intervening row separators or whitespace, instead you will first have to work out the size of the square grid based on the input and then use this value to determine the end of one row/column and the start of the next in your logic.

The `is_present` method should return `True` if `word` is present in the grid according to our rules or `False` otherwise.

To illustrate the pattern you can imagine that your class will be used as follows

```
ws = WordSearch(grid)
for word in words_to_find:
    if ws.is_present(word):
        print(f"found {word}")
```

Now the twist: what if the grid you will be given was a square grid with 10,000 ($10^4$) letters in each direction and `words_to_find` was an array of 1,000,000 ($10^6$) words. Imagine we're looking for a solution which is optimised for runtime efficiency. The sizes are not chosen for any specific magic reason, just to indicate that this is a "big" problem.

We're not looking for micro-optimisations in the code: if that were the case dispensing with Python altogether might be a better idea! Instead look at the algorithm side of things.

We will normally try to run the code under the latest stable Python 3 runtime, please consider code compatibility and document any particular restrictions.

As a bonus question (no need to implement), how would you go about taking advantage of a multicore system?