Daily Coding Problem #98

Problem

This problem was asked by Coursera.

Given a 2D board of characters and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example, given the following board:

```
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
```

`exists(board, "ABCCED")` returns `true`, `exists(board, "SEE")` returns `true`, `exists(board, "ABCB")` returns `false`.

Solution

We can view the provided board as a graph, where the characters are nodes, whose edges point to adjacent characters. We have the choice of performing a depth-first or breadth-first search. DFS is usually simpler to implement, so we will make that our choice. Also, since we can exit the search once we've reached the length of the word, the recursive depth will be limited by the length.

```
def search(board, row, col, word, index, visited):
    def is_valid(board, row, col):
        return row >= 0 and row < len(board) and col >= 0 and col < len(board[0])

    if not is_valid(board, row, col):
        return False
    if visited.contains((row, col)):
        return False
    if board[row][col] != word[index]:
        return False
    if index == len(word) - 1:
        return True

    visited.add((row, col))

    for d in ((0, -1), (0, 1), (-1, 0), (1, 0)):
        if search(board, row + d[0], col + d[1], word, index + 1):
            return True
```

```
    visited.remove((row, col))  # Backtrack

    return False

def find_word(board, word):
    int M = len(board)
    int N = len(board[0])

    for row in range(M):
        for col in range(N):
            visited = set()
            if search(board, row, col, word, 0, visited):
                return True
```

The worst-case time complexity of this solution is `O(MN * 4^L)` where `L` is the length of the word and M and N are the dimensions of the board.