Daily Coding Problem #23

Problem

This problem was asked by Google.

You are given an M by N matrix consisting of booleans that represents a board. Each True boolean represents a wall. Each False boolean represents a tile you can walk on.

Given this matrix, a start coordinate, and an end coordinate, return the minimum number of steps required to reach the end coordinate from the start. If there is no possible path, then return null. You can move up, left, down, and right. You cannot move through walls. You cannot wrap around the edges of the board.

For example, given the following board:

```
[[f, f, f, f],
[t, t, f, t],
[f, f, f, f],
[f, f, f, f]]
```

and start = `(3, 0)` (bottom left) and end = `(0, 0)` (top left), the minimum number of steps required to reach the end is 7, since we would need to go through `(1, 2)` because there is a wall everywhere else on the second row.

Solution

The idea here is to use either BFS or DFS to explore the board, starting from the start coordinate, and keep track of what we've seen so far as well as the steps from the start until we find the end coordinate.

In our case, we'll use BFS. We'll create a queue and initialize it with our start coordinate, along with a count of 0. We'll also initialize a `seen` set to ensure we only add coordinates we haven't seen before.

Then, as long as there's something still in the queue, we'll dequeue from the queue and first check if it's our target coordinate -- if it is, then we can just immediately return the count. Otherwise, we'll get the valid neighbours of the coordinate we're working with (valid means not off the board and not a wall), and enqueue them to the end of the queue.

To make sure the code doesn't get too messy, we'll define some helper functions: `walkable`, which returns whether or not a tile is valid, and `get_walkable_neighbours` which returns the valid neighbours of a coordinate.

```
from collections import deque

# Given a row and column, returns whether that tile is walkable.
def walkable(board, row, col):
    if row < 0 or row >= len(board):
        return False
    if col < 0 or col >= len(board[0]):
        return False
    return not board[row][col]
```

```
# Gets walkable neighbouring tiles.
def get_walkable_neighbours(board, row, col):
    return [(r, c) for r, c in [
        (row, col - 1),
        (row - 1, col),
        (row + 1, col),
        (row, col + 1)]
        if walkable(board, r, c)
    ]

def shortest_path(board, start, end):
    seen = set()
    queue = deque([(start, 0)])
    while queue:
        coords, count = queue.popleft()
        if coords == end:
            return count
        seen.add(coords)
        neighbours = get_walkable_neighbours(board, coords[0], coords[1])
        queue.extend((neighbour, count + 1) for neighbour in neighbours
                if neighbour not in seen)

board = [[False, False, False, False],
        [True, True, True, True],
        [False, False, False, False],
        [False, False, False, False]]

print(shortest_path(board, (3, 0), (0, 0)))
```

This code should run in O(M * N) time and space, since in the worst case we need to examine the entire board to find our target coordinate.