Daily Coding Problem #64

Problem

This problem was asked by Google.

A knight's tour is a sequence of moves by a knight on a chessboard such that all squares are visited once.

Given N, write a function to return the number of knight's tours on an N by N chessboard.

Solution

The brute force solution is here to try every possible permutation of moves and see if they're valid. That would be pretty much computationally infeasible, since we have N * N possible spots. That would be N^2!

We can improve the performance on this using backtracking, similar to the N queen problem (#38) or the flight itinerary problem (#41). The basic idea is this:

- For every possible square, initialize a knight there, and then:
- Try every valid move from that square.
- Once we've hit every single square, we can add to our count.

We'll represent the tour as just a sequence of tuples `(r, c)`. To speed things up and to avoid having to look at the whole tour to check whether a space has been used before, we can create an N by N board to mark whether we've seen it already.

```
def is_valid_move(board, move, n):
    r, c = move
    return 0 <= r < n and 0 <= c < n and board[r][c] is None

def valid_moves(board, r, c, n):
    deltas = [
        (2, 1),
        (1, 2),
        (1, -2),
        (-2, 1),
        (-1, 2),
        (2, -1),
        (-1, -2),
        (-2, -1),
    ]
    all_moves = [(r + r_delta, c + c_delta) for r_delta, c_delta in deltas]
    return [move for move in all_moves if is_valid_move(board, move, n)]

def knights_tours(n):
    count = 0
    for i in range(n):
        for j in range(n):
            board = [[None for _ in range(n)] for _ in range(n)]
            board[i][j] = 0
            count += knights_tours_helper(board, [(i, j)], n)
    return count

def knights_tours_helper(board, tour, n):
    if len(tour) == n * n:
        return 1
    else:
        count = 0
        last_r, last_c = tour[-1]
        for r, c in valid_moves(board, last_r, last_c, n):
            tour.append((r, c))
            board[r][c] = len(tour)
            count += knights_tours_helper(board, tour, n)
            tour.pop()
            board[r][c] = None
        return count
```

This takes O(N * N) space and potentially O(8^(N^2)) time, since at each step we have potentially 8 moves to check, and we have to do this for each square.