Daily Coding Problem #229

Problem

This problem was asked by Flipkart.

Snakes and Ladders is a game played on a `10 x 10` board, the goal of which is get from square `1` to square `100`. On each turn players will roll a six-sided die and move forward a number of spaces equal to the result. If they land on a square that represents a snake or ladder, they will be transported ahead or behind, respectively, to a new square.

Find the smallest number of turns it takes to play snakes and ladders.

For convenience, here are the squares representing snakes and ladders, and their outcomes:

```
snakes = {16: 6, 48: 26, 49: 11, 56: 53, 62: 19, 64: 60, 87: 24, 93: 73, 95: 75, 98: 78}
ladders = {1: 38, 4: 14, 9: 31, 21: 42, 28: 84, 36: 44, 51: 67, 71: 91, 80: 100}
```

Solution

We know that during each turn a player has six possible moves, advancing from one to six squares. So our first thought might be to recursively try each move, exploring all possible paths. Then, we can return the length of the shortest path.

However, because there are snakes, we would eventually enter a never-ending loop, repeatedly advancing to a snake square and being sent back. Even without this issue, this solution would be exponentially slow, since we have six potential moves at each square.

A more efficient method is to use a version of breadth-first search.

We can maintain a queue of tuples representing the current square and the number of turns taken so far, starting with `(0, 0)`. For each item popped from the queue, we examine the moves that can be made from it. If a move crosses the finish line, we've found a solution. Otherwise, if a move takes us to a square we have not already visited, we add that square to the queue.

The key point here is that squares will only be put in the queue on the earliest turn they can be reached. For example, even though it is possible to reach square 5 by moving `[1, 1, 1, 1, 1]`, the initial move 5 will get there first. So we can guarantee that we will only examine each square once, and that the number of turns associated with each square will be minimal.

```
from collections import deque

def minimum_turns(board):
    start, end = 0, 100
    turns = 0
    path = deque([(start, turns)])
    visited = set()

    while path:
        square, turns = path.popleft()

        for move in range(square + 1, square + 7):
            if move >= end:
```

```
                return turns + 1

            if move not in visited:
                visited.add(move)
                path.append((board[move], turns + 1))

assert minimum_turns(board) == 7
```

Since each square is only placed in the queue once, and our queue operations take constant time, this algorithm is linear in the number of squares.
