Daily Coding Problem #281

Problem

This problem was asked by LinkedIn.

A wall consists of several rows of bricks of various integer lengths and uniform height. Your goal is to find a vertical line going from the top to the bottom of the wall that cuts through the fewest number of bricks. If the line goes through the edge between two bricks, this does not count as a cut.

For example, suppose the input is as follows, where values in each row represent the lengths of bricks in that row:

```
[[3, 5, 1, 1],
 [2, 3, 3, 2],
 [5, 5],
 [4, 4, 2],
 [1, 3, 3, 3],
 [1, 1, 6, 1, 1]]
```

The best we can we do here is to draw a line after the eighth brick, which will only require cutting through the bricks in the third and fifth row.

Given an input consisting of brick lengths for each row such as the one above, return the fewest number of bricks that must be cut to create a vertical line.

Solution

At first glance we might consider testing each vertical line to see how many bricks it

would have to cut through. However, given the structure of our input, each line will require us to accumulate the values of each row of bricks, which will be both messy and time-consuming. If the length of the wall is `M` and there are `N` total bricks, this will take `O(M * N)`.

For problems that involve optimization, a little reframing can often help. Instead of thinking about how to minimize the number of cuts, we can try to maximize the number of times a line can pass through an edge between two bricks.

To do this, we can examine each row and increment a counter in a hash map for the accumulated distance covered after each brick, except for the last one. For example, after the first row in the input above, our map would contain `{3: 1, 8: 1, 9: 1}`. The key in the dictionary with the largest value represents the vertical line with the fewest bricks cut. Finally, to find the actual number of bricks, we can subtract this value from the total number of rows.

```
from collections import defaultdict

def fewest_cuts(wall):
    cuts = defaultdict(int)

    for row in wall:
        length = 0
        for brick in row[:-1]:
            length += brick
            cuts[length] += 1

    return len(wall) - max(cuts.values())
```

For each brick, we only need to update our dictionary once, so this algorithm will take `O(N)` time. Our map will require `O(M)` space to store a value for each possible edge.