Documentation for Brick Wall Problem

📚 Table of Contents

1. Problem Statement

You're given a rectangular brick wall represented as a 2D list. Each sub-list represents a row of bricks with variable widths. All rows sum to the same total width.

Your task is to draw a vertical line from top to bottom that crosses the fewest number of bricks. The line can only pass through the gaps between bricks, not just along the two vertical wall edges.

Return the minimum number of bricks crossed by such a line.

2. 💡 Intuition

We can minimize the number of bricks crossed by drawing the vertical line through a position that has the most aligned gaps. Every gap shared between rows is an opportunity to avoid crossing a brick.

3. 🔍 Key Observations

- Gaps between bricks can be represented as cumulative sums of brick widths.
- The line cannot be drawn along the rightmost edge of the wall.
- If the line passes through a common edge position, it doesn't count as crossing a brick.

4. □ Approach

- Iterate through each row.
- For each brick (excluding the last), compute the cumulative width and store the count of that edge position in a dictionary.
- Find the maximum frequency of any edge position (i.e., the position where most gaps align).
- Subtract this count from the total number of rows to get the minimum number of bricks crossed.

5. □ Edge Cases

- Wall with only one brick per row: every line will cross all bricks.
- No repeated edge positions: line will cross all rows.
- Empty input: return 0.
- All rows have the same pattern: minimum crossings will be optimal.

6. □ Complexity Analysis

Time Complexity

$O(N \times M)$
Where N is the number of rows and M is the average number of bricks in a row.

Space Complexity

$O(W)$
Where W is the wall width (number of unique edge positions).

7. 🔁 Alternative Approaches

- Brute Force: Try every possible vertical position between 1 and wall width - 1 and count how many bricks each line crosses (inefficient).
- Prefix Sum Array for Each Row: Similar to the optimal approach but consumes more memory.

8. 📄 Algorithm

- Initialize a hashmap to store edge positions and their frequency.
- For each row:
    - Calculate the prefix sum excluding the last brick.
    - Increment the count for each edge position.
- Get the position with the maximum frequency.
- Return number of rows - max frequency.

9. □ Test Cases

✅ Example 1

Input: [[1,2,2,1],[3,1,2],[1,3,2],[2,4],[3,1,2],[1,3,1,1]]
Output: 2

✅ Example 2

Input: [[1],[1],[1]]
Output: 3

✅ Example 3

Input: [[1,1],[2],[1,1]]
Output: 1

10. □ Final Thoughts

This problem showcases the power of prefix sums and hash maps in detecting common patterns. It avoids brute-force checking of every possible vertical line and instead leverages gap alignment across rows for an efficient solution.