**Problem Statement: The Wall**

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.

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

**Input Format**

* The first line contains an integer N, the number of rows in the wall.
* The next N lines each start with an integer k representing the number of bricks in the row, followed by k space-separated integers representing the lengths of bricks in that row.

**Output Format**

* An integer representing the minimum number of bricks that must be cut.

**Constraints**

* Each row of the wall will have at least one brick.
* The lengths of bricks in each row are positive integers.

**Example**

**Input:**

6

5 4 3 5 1 1

5 4 2 3 3 2

3 2 5 5

4 3 4 4 2

4 1 3 3 3

5 1 1 6 1 1

**Output:**

2

**Explanation:**

**Explanation**

The input represents the following wall:

* Row 1: [4, 3, 5, 1, 1]
* Row 2: [4, 2, 3, 3, 2]
* Row 3: [2, 5, 5]
* Row 4: [3, 4, 4, 2]
* Row 5: [1, 3, 3, 3]
* Row 6: [1, 1, 6, 1, 1]

We need to find a vertical line that cuts through the fewest number of bricks.

To solve this, we calculate the positions where the bricks end for each row:

* Row 1: [4, 7, 12, 13]
* Row 2: [4, 6, 9, 12]
* Row 3: [2, 7]
* Row 4: [3, 7, 11]
* Row 5: [1, 4, 7]
* Row 6: [1, 2, 8, 9]

We then count the number of edges that occur at each position (except the last edge since it’s the end of the wall). The position with the highest count of edges will be our optimal position for the vertical line, as it will pass through the least number of bricks.

Here's a step-by-step breakdown:

1. Calculate the prefix sums (positions where bricks end) for each row.
2. Create a dictionary to count how many times each position appears across all rows.
3. The position with the highest count in the dictionary is the position where we should draw the vertical line to minimize cuts.

Using the counts from the example:

* Position 4: appears in Row 1, Row 2, Row 4
* Position 7: appears in Row 1, Row 3, Row 4, Row 5

The optimal position is 7, as it has the highest count (4). This means the vertical line will cut through the fewest number of bricks at position 7. All other positions have fewer counts.

Hence, the fewest number of bricks that need to be cut is 6−4=26 - 4 = 26−4=2.

**Solution**

To solve this problem, we can follow these steps:

1. Calculate the prefix sums of the brick lengths for each row.
2. Use a dictionary to count the occurrences of each prefix sum.
3. The key with the highest count will be the position where the vertical line can pass through the maximum number of brick edges, resulting in the fewest cuts.
4. The number of cuts will be the total number of rows minus the maximum count found in the dictionary.

Here is the implementation:

python

Copy code

def fewest\_bricks\_to\_cut(wall):

from collections import defaultdict

edge\_count = defaultdict(int)

# Calculate prefix sums for each row, except the last brick in each row

for row in wall:

prefix\_sum = 0

for brick in row[:-1]:

prefix\_sum += brick

edge\_count[prefix\_sum] += 1

# Find the position with the maximum number of edges

max\_edges = max(edge\_count.values(), default=0)

# The minimum cuts will be the total number of rows minus the max\_edges

return len(wall) - max\_edges

# Test the function with the given example

if \_\_name\_\_ == "\_\_main\_\_":

import sys

input = sys.stdin.read

data = input().split()

N = int(data[0])

wall = []

index = 1

for \_ in range(N):

k = int(data[index])

index += 1

row = []

for \_ in range(k):

row.append(int(data[index]))

index += 1

wall.append(row)

print(fewest\_bricks\_to\_cut(wall))

**Test Cases**

**Test Case 1:**

**Input:**

6

5 4 3 5 1 1

5 4 2 3 3 2

3 2 5 5

4 3 4 4 2

4 1 3 3 3

5 1 1 6 1 1

**Output:**

2

**Test Case 2:**

**Input:**

4

4 1 2 2 1

4 2 3 2

3 1 2 1

2 2 4

**Output:**

2

**Test Case 3:**

**Input:**

3

1 1

1 1

1 1

**Output:**

3

**Test Case 4:**

**Input:**

3

2 3 3

2 3 3

2 3 3

**Output:**

0

**Test Case 5:**

**Input:**

6

4 1 2 2 1

4 2 3 2

4 1 2 1 1

2 2 4

4 3 1 1 2

3 1 1 6

**Output:**

3