There is a brick wall in front of you. The wall is rectangular and has several rows of bricks. The bricks have the same height but different width. You want to draw a vertical line from the **top** to the **bottom** and cross the **least** bricks.

The brick wall is represented by a list of rows. Each row is a list of integers representing the width of each brick in this row from left to right.

If your line go through the edge of a brick, then the brick is not considered as crossed. You need to find out how to draw the line to cross the least bricks and return the number of crossed bricks.

You cannot draw a line just along one of the two vertical edges of the wall, in which case the line will obviously cross no bricks.

Example:

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

Output: 2

Explanation:
```

Note:

1. The width sum of bricks in different rows are the same and won't exceed INT_MAX.
2. The number of bricks in each row is in range [1,10,000]. The height of wall is in range [1,10,000]. Total number of bricks of the wall won't exceed 20,000.

Intution:

1. we will keep summing the width of each brick for every row
2. for ex for the first level = [1 3 5 6]
3. final sum is ofcourse going to be same (width of the wall) for each row(in this case : 6)
4. We know if we cross the edge of a brick, it is not counted
5. Also if we are getting a sum S at some position in one row, and same sum S in the below row, this means that they both are having edge at the same position
6. for example for S = 4, [1 3] and [3 1] are having edge at same point => it will be profitable to cross the wall along the path where max no of rows are having edge at the same common point
7. In other words, we should form the line along the path where we have the most common prefix sum
8. we will count the number of rows with the most common prefix sum and subtract this number from size of the wall to get the final answer.

CODE:

```
class Solution {
```

```
public:
    int leastBricks(vector<vector<int>>& wall) {
        map<int,int> m;
        for(auto v : wall){
            int sum = 0;
            for(int i=0;i<v.size()-1;i++){
                sum += v[i];
                m[sum]++;
            }
        }
        int mc=0;
        for(auto it : m){
            if(mc < it.second)
                mc = it.second;
        }
        return wall.size() - mc;
    }
};
```

NOTE :
In order to speedify your solution on any platform you can add this peice of code above the class Solution

```
static int accel = []() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    return 0;
}();
```