**CSCE 5150: Problem of the Day**

**Name: Kishan Kumar Zalavadia**

**Question:**

Just imagining that you have n super washing machines on a line. Initially, each washing machine has some dresses or is empty.

For each move, you could choose any m (1 <= m <= n) washing machines and pass one dress of each washing machine to one of its adjacent washing machines at the same time.

Given an integer array of machines representing the number of dresses in each washing machine from left to right on the line, return the minimum number of moves to make all the washing machines have the same number of dresses. If it is not possible to do it, return -1.

**Example 1:**

Input: machines = [1,0,5] Output: 3 Explanation: 1st move: 1 0 <-- 5 => 1 1 4 2nd move: 1 <-- 1 <-- 4 => 2 1 3 3rd move: 2 1 <-- 3 => 2 2 2

**Example 2:**

Input: machines = [0,3,0] Output: 2 Explanation: 1st move: 0 <-- 3 0 => 1 2 0 2nd move: 1 2 --> 0 => 1 1 1

**Example 3:**

Input: machines = [0,2,0] Output: -1 Explanation: It's impossible to make all three washing machines have the same number of dresses.

**Solution:**

1. **int n = machines.length;**: Get the number of washing machines in the array.
2. Calculate the total number of dresses across all machines:
   * **int totalDresses = 0;**: Initialize a variable to keep track of the total number of dresses.
   * **for (int dresses : machines) { totalDresses += dresses; }**: Iterate through the **machines** array and sum up the number of dresses in each machine.
3. Check if the total number of dresses can be evenly distributed among the machines:
   * **if (totalDresses % n != 0) { return -1; }**: If the total number of dresses divided by the number of machines (**n**) leaves a remainder, it's impossible to distribute the dresses evenly, so return -1.
4. Calculate the target number of dresses that each machine should have:
   * **int targetDresses = totalDresses / n;**: Divide the total number of dresses by the number of machines to find the target number of dresses for each machine.
5. Initialize variables for tracking moves and balance:
   * **int moves = 0;**: Initialize a variable to keep track of the number of moves needed to balance the dresses.
   * **int balance = 0;**: Initialize a variable to keep track of the excess (positive) or shortage (negative) of dresses.
6. Iterate through the machines to calculate the moves needed to balance them:
   * **for (int dresses : machines) {**: Loop through each machine's dress count.
   * **int diff = dresses - targetDresses;**: Calculate the difference between the current number of dresses in the machine and the target number.
   * **balance += diff;**: Accumulate the difference to the **balance** variable, which keeps track of the total excess or shortage of dresses.
   * **moves = Math.max(moves, Math.max(Math.abs(balance), diff));**: Update the **moves** variable to be the maximum of the current **moves**, the absolute value of **balance**, and **diff**. This ensures that we choose the maximum number of moves needed to balance the dresses.
7. Finally, return the calculated **moves**, which represents the minimum number of moves needed to balance the dresses in the washing machines.

**Code:**

class Solution {

public int findMinMoves(int[] machines) {

int n = machines.length;

// Calculate the total number of dresses across all machines

int totalDresses = 0;

for (int dresses : machines) {

totalDresses += dresses;

}

// If the total number of dresses cannot be evenly distributed, return -1

if (totalDresses % n != 0) {

return -1;

}

// Calculate the target number of dresses that each machine should have

int targetDresses = totalDresses / n;

int moves = 0; // Initialize the number of moves needed

// Initialize balance variable to keep track of excess or shortage

int balance = 0;

// Iterate through machines to calculate moves needed to balance them

for (int dresses : machines) {

// Calculate the excess (positive) or shortage (negative) of dresses

int diff = dresses - targetDresses;

// Accumulate the difference to the balance

balance += diff;

// The maximum of balance and diff is the actual moves needed

moves = Math.max(moves, Math.max(Math.abs(balance), diff));

}

return moves;

}

}