Super Washing Machines

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 representing the number of dresses in each washing machine from left to right on the line, you should find 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.

Example1

```
Input: [1,0,5]

Output: 3

Explanation:
1st move:    1     0     1     1     4
2nd move:    1     2     1     3
3rd move:    2     1     2     2     2
```

Example2

```
Input: [0,3,0]

Output: 2

Explanation:
1st move:    0     1     2     0
2nd move:    1     2 --> 0    =>    1     1     1
```

Example3

```
Input: [0,2,0]

Output: -1

Explanation:
It's impossible to make all the three washing machines have the same number of dres
ses.
```

Note:

1. The range of n is [1, 10000].
2. The range of dresses number in a super washing machine is [0, 1e5].

Solution 1

First we check the sum of dresses in all machines. if that number cannot be divided by count of machines, there is no solution.

Otherwise, we can always transfer a dress from one machine to another, one at a time until every machines reach the same number, so there must be a solution. In this way, the total actions is sum of operations on every machine.

Since we can operate several machines at the same time, the minium number of moves is the maximum number of necessary operations on every machine.

For a single machine, necessary operations is to transfer dresses from one side to another until sum of both sides and itself reaches the average number. We can calculate (required dresses) - (contained dresses) of each side as L and R:

L > 0 && R > 0: both sides lacks dresses, and we can only export one dress from current machines at a time, so result is abs(L) + abs(R)
L < 0 && R < 0: both sides contains too many dresses, and we can import dresses from both sides at the same time, so result is max(abs(L), abs(R))
L < 0 && R > 0 or L >0 && R < 0: the side with a larger absolute value will import/export its extra dresses from/to current machine or other side, so result is max(abs(L), abs(R))

For example, [1, 0, 5], average is 2
for 1, L = 0 * 2 - 0 = 0, R = 2 * 2 - 5= -1, result = 1
for 0, L = 1 * 2 - 1= 1, R = 1 * 2 - 5 = -3, result = 3
for 5, L = 2 * 2 - 1= 3, R = 0 * 2 - 0= 0, result = 3
so minium moves is 3

```cpp
class Solution {
public:
    int findMinMoves(vector<int>& machines) {
        int len = machines.size();
        vector<int> sum(len + 1, 0);
        for (int i = 0; i < len; ++i)
            sum[i + 1] = sum[i] + machines[i];

        if (sum[len] % len) return -1;

        int avg = sum[len] / len;
        int res = 0;
        for (int i = 0; i < len; ++i)
        {
            int l = i * avg - sum[i];
            int r = (len - i - 1) * avg - (sum[len] - sum[i] - machines[i]);

            if (l > 0 && r > 0)
                res = std::max(res, std::abs(l) + std::abs(r));
            else
                res = std::max(res, std::max(std::abs(l), std::abs(r)));
        }
        return res;
    }
};
```

written by Mrsuyi original link here

Solution 2

```
public class Solution {
    public int findMinMoves(int[] machines) {
        int total = 0;
        for(int i: machines) total+=i;
        if(total%machines.length!=0) return -1;
        int avg = total/machines.length, cnt = 0, max = 0;
        for(int load: machines){
            cnt += load-avg; //load-avg is "gain/lose"
            max = Math.max(Math.max(max, Math.abs(cnt)), load-avg);
        }
        return max;
    }
}
```

Let me use an example to briefly explain this. For example, your machines[] is [0,0,11,5]. So your total is 16 and the target value for each machine is 4. Convert the machines array to a kind of gain/lose array, we get: [-4,-4,7,1]. Now what we want to do is go from the first one and try to make all of them 0.
To make the 1st machines 0, you need to give all its "load" to the 2nd machines.
So we get: [0,-8,7,1]
then: [0,0,-1,1]
lastly: [0,0,0,0], done.
You don't have to worry about the details about how these machines give load to each other. In this process, the least steps we need to eventually finish this process is determined by the peak of abs(cnt) and the max of "gain/lose" array. In this case, the peak of abs(cnt) is 8 and the max of gain/lose array is 7. So the result is 8.

Some other example:
machines: [0,3,0]; gain/lose array: [-1,2,-1]; max = 2, cnt = 0, -1, 1, 0, its abs peak is 1. So result is 2.
machines: [1,0,5]; gain/lose array: [-1,-2,3]; max = 3, cnt = 0, -1, -3, 0, its abs peak is 3. So result is 3.

written by Chidong original link here

Solution 3

```
public class Solution {
    public int findMinMoves(int[] machines) {
        int total = 0, target = 0, result = 0, n = machines.length;
        for (int d : machines) total += d;
        if (total == 0) return 0;
        if (total % n != 0) return -1;
        target = total / n;

        int[] move = new int[n];
        for (int i = 0; i < n - 1; i++) {
            if (machines[i] > target) {
                move[i] += machines[i] - target;
                machines[i + 1] += machines[i] - target;
                machines[i] = target;
                result = Math.max(result, move[i]);
            }
            else {
                move[i + 1] = target - machines[i];
                machines[i + 1] -= target - machines[i];
                machines[i] = target;
                result = Math.max(result, move[i + 1]);
            }
        }

        return result;
    }
}
```

written by shawngao original link here