

517. Super washing machines

May 31, 2019 | 10K views

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 ($1 \leq m \leq n$) 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 <-- 5 => 1 1 4
2nd move: 1 <-- 1 <-- 4 => 2 1 3
3rd move: 2 1 <-- 3 => 2 2 2

Example2

Input: [0,3,0]

Output: 2

Explanation:
1st move: 0 <-- 3 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 dresses.

Note:

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

Solution

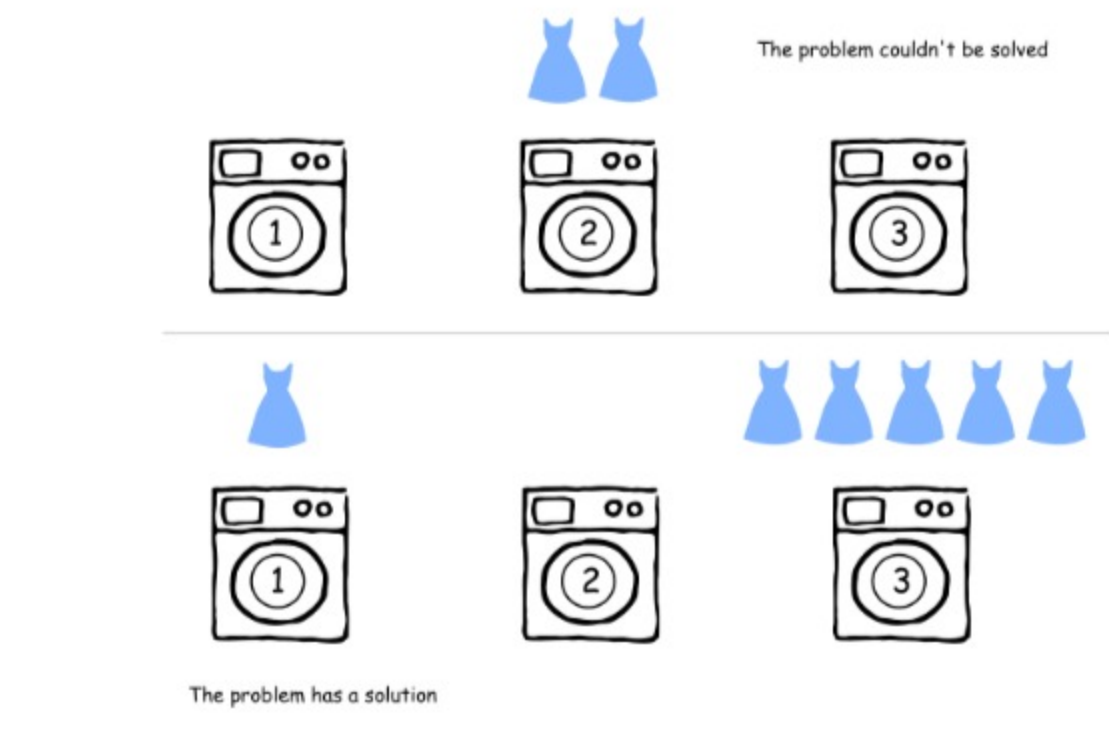
Approach 1: Greedy.

Intuition

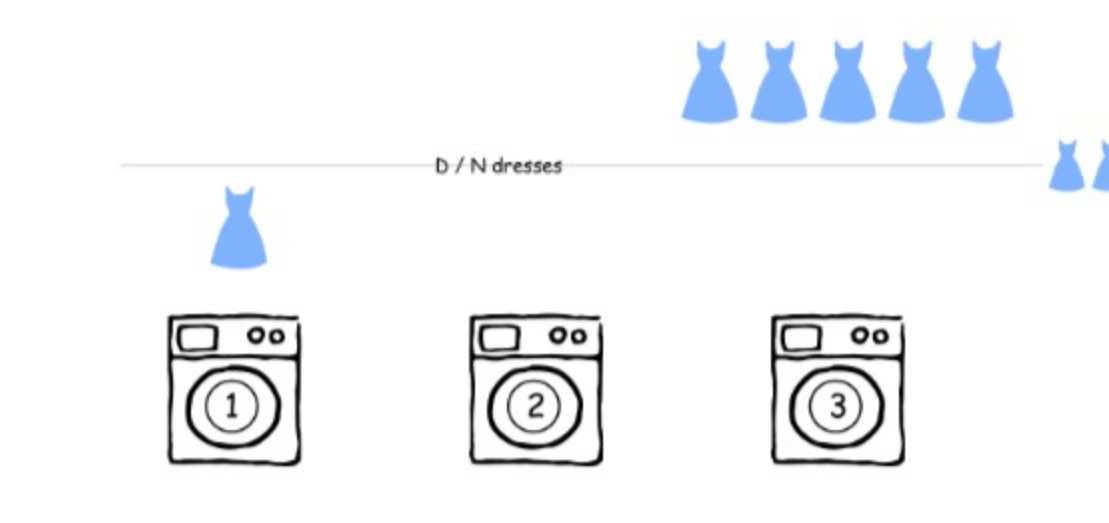
This greedy problem is very similar to [Gas station problem](#), and could be solved in linear time as well.

First of all - could the problem be solved or not?

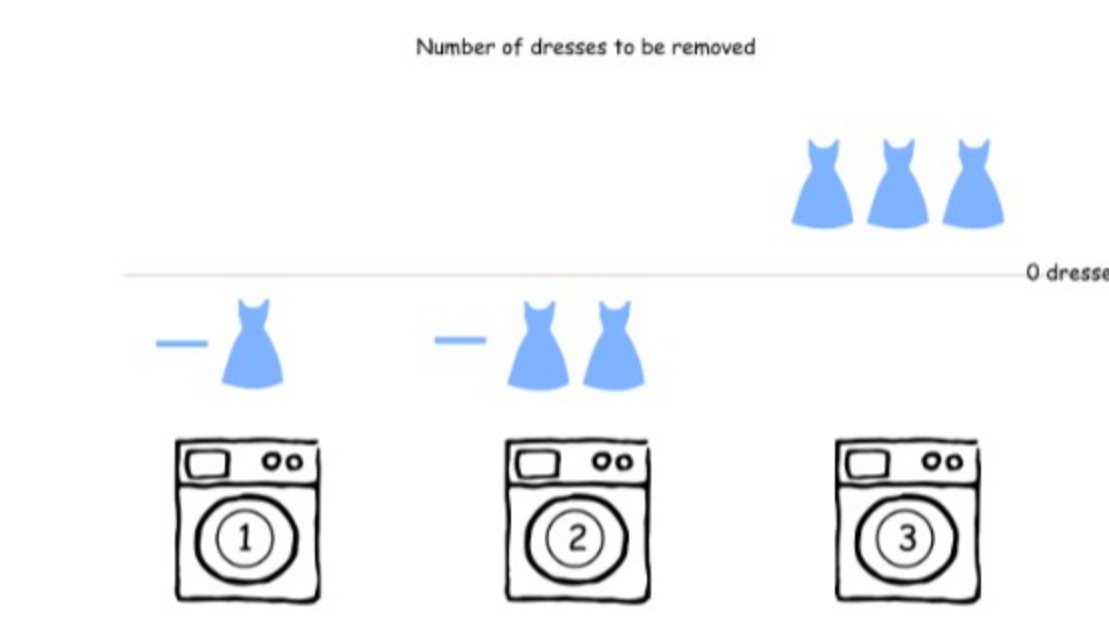
Yes, if the dresses could be divided into N equal parts where N is number of machines. In other words, N should be a divisor of the number of dresses D .



Now it's easy to compute the number of dresses that each machine should have: D / N . The starting numbers of dresses in the machines move around this D / N average value.



The standard ML trick is to normalize the data, so that the average value would be zero. For that, one could replace the actual number of dresses in the machine by the number of dresses to be removed. This number could be negative, if one actually needs to *add* the dresses into the machine.



As for the [gas station problem](#), one starts from the beginning and checks the standard set for such problems: the current element, the current sum, and the maximum sum seen so far:

- m . Number of dresses to be removed from the current machine.
- $curr_sum$. Number of dresses to be passed on the right.
- max_sum . Maximum number of dresses one had to pass on the right at this point or before.

It's quite obvious that the result at each point is a maximum between max_sum and m , i.e. one has to compare the cumulative and the local maximums.

Here are three different examples.

- $[1, 0, 5]$. The cumulative maximum is equal to the local one.

<code>result = max(result, max_sum, m)</code>	1	3	3
<code>max_sum</code> = Max number of dresses one had to pass on the right so far	1	3	3
<code>curr_sum</code> = Number of dresses to be passed on the right	-1	-3	0
<code>m</code> = Number of dresses to be removed from this machine	-1	-2	-1
Washing machine	1	2	3

- $[0, 3, 0]$. The local maximum wins over the cumulative one.

<code>result = max(result, max_sum, m)</code>	1	2	2
<code>max_sum</code> = Max number of dresses one had to pass on the right so far	1	1	1
<code>curr_sum</code> = Number of dresses to be passed on the right	-1	1	0
<code>m</code> = Number of dresses to be removed from this machine	-1	-2	-1
Washing machine	1	2	3

- $[0, 0, 3, 5]$. The cumulative maximum wins over the local one.

<code>result = max(result, max_sum, m)</code>	2	4	4	4
<code>max_sum</code> = Max number of dresses one had to pass on the right so far	2	4	4	4
<code>curr_sum</code> = Number of dresses to be passed on the right	-2	-4	-3	0
<code>m</code> = Number of dresses to be removed from this machine	-2	-3	-2	-1
Washing machine	1	2	3	4

Algorithm

Here is the algorithm.

- Check if the problem could be solved: $len(machines)$ should be a divisor of $sum(machines)$. Otherwise the answer is -1.
- Compute the number of dresses each machine should finally have: $dresses_per_machine = sum(machines)/len(machines)$.
- Normalize** the problem by replacing the *number of dresses* in each machine by the *number of dresses to be removed* from this machine (could be negative).
- Initiate `curr_sum`, `max_sum`, and `res` as zero.
- Iterate over all machines `m` in `machines`:
 - Update `curr_sum` and `max_sum` at each step: `curr_sum += m`, `max_sum = max(max_sum, abs(curr_sum))`.
 - Update result `res = max(res, max_sum, m)`.
- Return `res`.

Implementation

JavaPython

Copy

```
1 class Solution:
2     def findMinMoves(self, machines: List[int]) -> int:
3         n = len(machines)
4         dress_total = sum(machines)
5         if dress_total % n != 0:
6             return -1
7
8         dress_per_machine = dress_total // n
9         for i in range(n):
10             # Change the number of dresses in the machines to
11             # the number of dresses to be removed from this machine
12             # (could be negative)
13             machines[i] -= dress_per_machine
14
15         # curr_sum is a number of dresses to move at this point,
16         # max_sum is a max number of dresses to move at this point or before,
17         # m is number of dresses to move out from the current machine.
18         curr_sum = max_sum = res = 0
19         for m in machines:
20             curr_sum += m
21             max_sum = max(max_sum, abs(curr_sum))
22             res = max(res, max_sum, m)
23         return res
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N)$ since it's a three iterations over the input array.
- Space complexity: $\mathcal{O}(1)$ since it's a constant space solution.

Rate this article: ★★★★★

PreviousNext

Comments: 9

Sort By

🔊

Type comment here... (Markdown is supported)

PreviewPost

👤 ping_pong

★ 628

🕒 August 7, 2019 9:17 PM

Solution doesn't appear very intuitive to me, can anyone provide better explanation.

22 🗳️ 🗳️ 🗳️ | 🗳️ Share | 🗳️ Reply

👤 vTray

★ 16

🕒 March 13, 2020 12:51 PM

Terrible explanation. Couldn't understand a word.

7 🗳️ 🗳️ 🗳️ | 🗳️ Share | 🗳️ Reply

👤 Primusai

★ 224

🕒 August 18, 2019 8:03 PM

FYI computing `max_sum` isn't necessary here - it's kinda confusing and doesn't accomplish anything. You can replace the bottom part with:

```
curr_sum = res = 0
for m in machines:
    curr_sum += m
```

7 🗳️ 🗳️ 🗳️ | 🗳️ Share | 🗳️ Reply

SHOW 1 REPLY

👤 DCXiaoBing

★ 55

🕒 June 20, 2019 2:51 AM

Nice problem.

1 🗳️ 🗳️ 🗳️ | 🗳️ Share | 🗳️ Reply

👤 Cool97

★ 1

🕒 June 2, 2019 7:25 PM

Gas Station Problem link is not working. Shouldn't the answer be sum of all positive numbers after normalizing? All the above examples work for this hypothesis. Can you please give me an example where it doesn't work for this hypothesis?

1 🗳️ 🗳️ 🗳️ | 🗳️ Share | 🗳️ Reply

SHOW 2 REPLIES

👤 XirenZhou

★ 5

🕒 May 27, 2020 6:35 AM

If you want a detailed explanation (rigorous proof) of why the solution works, please refer to this post: <https://leetcode.com/problems/super-washing-machines/discuss/654317/Explanation-proof-of-why-the-solution-works>.

0 🗳️ 🗳️ 🗳️ | 🗳️ Share | 🗳️ Reply

👤 qvani

★ 0

🕒 April 21, 2020 2:29 AM

Explanation looks much more complicated than below two scans solution. Does it have any missing cases?

```
int superWashingMachine(int* arr, int arrSize)
{
    // ...
}
```

0 🗳️ 🗳️ 🗳️ | 🗳️ Share | 🗳️ Reply

SHOW 2 REPLIES

👤 Tegegnem

★ 16

🕒 March 8, 2020 5:01 AM

res = Math.max(Math.max(Math.abs(currSum-m),m),res);

the idea is to compare m (the current value of machines) with the absolute value of the the running sum...the above expression does that...no temp variables.

0 🗳️ 🗳️ 🗳️ | 🗳️ Share | 🗳️ Reply

👤 liutinglin86

★ 0

🕒 December 31, 2019 1:41 PM

Shouldn't also add abs() on m when computing res? like

```
curr_sum = res = 0
for m in machines:
    curr_sum += m
```

0 🗳️ 🗳️ 🗳️ | 🗳️ Share | 🗳️ Reply