Previous Next

517. Super washing machines

May 31, 2019 | 10K views

Average Rating: 2.59 (22 votes)

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** ($1 \le m \le n$) 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 dresse
```

Note:

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

Solution

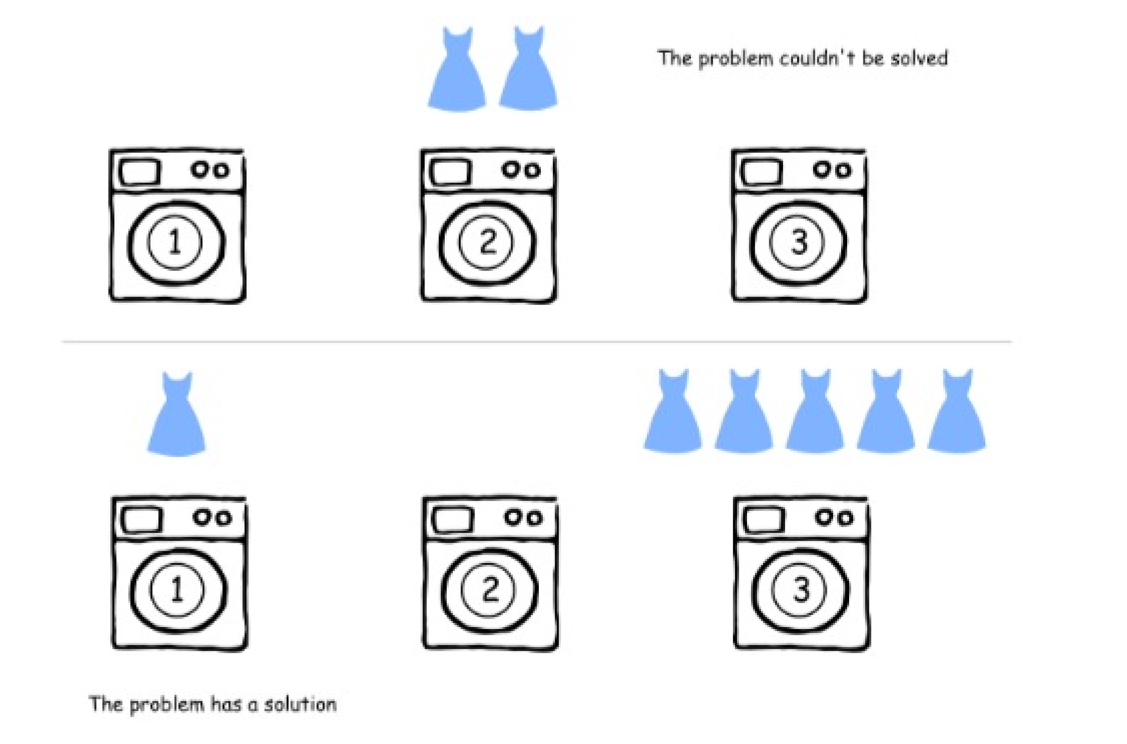
Approach 1: Greedy.

Intuition

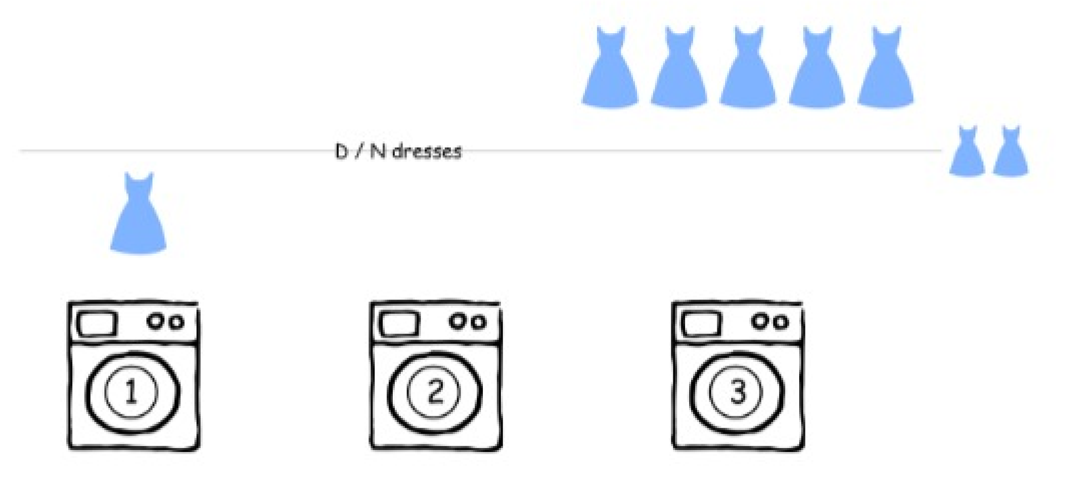
This greedy problem is very similar to Gas station problem, and could be solved in linear time as well.

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

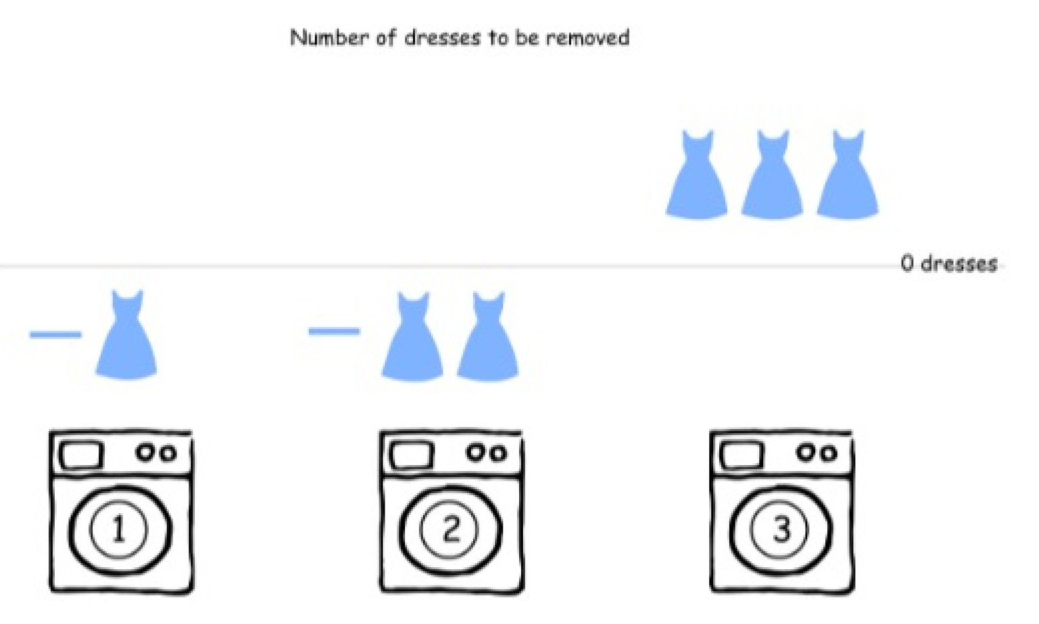
Yes, if the dresses could be divided into `N` equal parts where `N` is number of machines. In other words, `N` should be a divisor of the number of dresses `D`.



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.

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

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

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

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

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

Algorithm

Here is the algorithm.

1. Check if the problem could be solved: `len(machines)` should be a divisor of `sum(machines)`. Otherwise the answer is `-1`.
2. Compute the number of dresses each machine should finally have: `dresses_per_machine = sum(machines)/len(machines)`.
3. 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).
4. Initiate `curr_sum`, `max_sum`, and `res` as zero.
5. 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)`.
6. Return `res`.

Implementation

Java | Python | Copy

```python
class Solution:
    def findMinMoves(self, machines: List[int]) -> int:
        n = len(machines)
        dress_total = sum(machines)
        if dress_total % n != 0:
            return -1

        dress_per_machine = dress_total // n
        for i in range(n):
            # Change the number of dresses in the machines to
            # the number of dresses to be removed from this machine
            # (could be negative)
            machines[i] -= dress_per_machine

        # curr_sum is a number of dresses to move at this point,
        # max_sum is a max number of dresses to move at this point or before,
        # m is number of dresses to move out from the current machine.
        curr_sum = max_sum = res = 0
        for m in machines:
            curr_sum += m
            max_sum = max(max_sum, abs(curr_sum))
            res = max(res, max_sum, m)
        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:

Previous Next