

[Advent of Code](#) [\[About\]](#) [\[AoC++\]](#) [\[Events\]](#) [\[Settings\]](#) [\[Log Out\]](#) [Shinjaryuu 2*](#)
[{year=>2017}](#) [\[Calendar\]](#) [\[Leaderboard\]](#) [\[Stats\]](#) [\[Sponsors\]](#)

--- Day 6: Memory Reallocation ---

A debugger program here is having an issue: it is trying to repair a memory reallocation routine, but it keeps getting stuck in an infinite loop.

In this area, there are sixteen memory banks; each memory bank can hold any number of blocks. The goal of the reallocation routine is to balance the blocks between the memory banks.

The reallocation routine operates in cycles. In each cycle, it finds the memory bank with the most blocks (ties won by the lowest-numbered memory bank) and redistributes those blocks among the banks. To do this, it removes all of the blocks from the selected bank, then moves to the next (by index) memory bank and inserts one of the blocks. It continues doing this until it runs out of blocks; if it reaches the last memory bank, it wraps around to the first one.

The debugger would like to know how many redistributions can be done before a blocks-in-banks configuration is produced that has been seen before.

For example, imagine a scenario with only four memory banks:

- The banks start with `0`, `2`, `7`, and `0` blocks. The third bank has the most blocks, so it is chosen for redistribution.
- Starting with the next bank (the fourth bank) and then continuing to the first bank, the second bank, and so on, the `7` blocks are spread out over the memory banks. The fourth, first, and second banks get two blocks each, and the third bank gets one back. The final result looks like this: `2 4 1 2`.
- Next, the second bank is chosen because it contains the most blocks (four). Because there are four memory banks, each gets one block. The result is: `3 1 2 3`.
- Now, there is a tie between the first and fourth memory banks, both of which have three blocks. The first bank wins the tie, and its three blocks are distributed evenly over the other three banks, leaving it with none: `0 2 3 4`.
- The fourth bank is chosen, and its four blocks are distributed such that each of the four banks receives one: `1 3 4 1`.
- The third bank is chosen, and the same thing happens: `2 4 1 2`.

At this point, we've reached a state we've seen before: `2 4 1 2` was already seen. The infinite loop is detected after the fifth block redistribution cycle, and so the answer in this example is `5`.

Given the initial block counts in your puzzle input, how many redistribution cycles must be completed before a configuration is produced that has been seen before?

To begin, [get your puzzle input](#).

Answer: [\[Submit\]](#)

You can also [\[Share\]](#) this puzzle.

Our [sponsors](#) help make Advent of Code possible:

[Novetta](#) - Unleash your imagination. Innovate at Novetta.

Deep Learning mit MATLAB

Deep Learning eBook herunterladen und gratis 30 Tage MATLAB Deep Learning testen

MathWorks

