# Perfect Deck

Atul Dhingra

August 27, 2017

## Problem Statement

You are given a deck containing N cards, while holding the deck:

1. Take the top card off the deck and set it on the table
2. Take the next card off the top and put it on the bottom of the deck in your hand.
3. Continue steps 1 and 2 until all cards are on the table. This is a round.
4. Pick up the deck from the table and repeat steps 1-3 until the deck is in the original order.

Write a program to determine how many rounds it will take to put a deck back into the original order. The program should take the number of cards in the deck as a command line argument and write the result to stdout.

## Analysis

The naive solution to this problem is to treat the the card deck as an array and go through steps 1 to step 3 until the original configuration of the array is obtained. This approach is tractable where the number of cards in the deck is small, but as the number of cards in the deck increase, the run time increases.

### Intuition

Consider the following example, where the deck is of size 11( cards numbered between 0 and 10). Following is the status of the deck after each round,

$$round_0 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

$$round_1 = [5, 9, 1, 7, 3, 10, 8, 6, 4, 2, 0]$$

$$round_2 = [10, 2, 9, 6, 7, 0, 4, 8, 3, 1, 5]$$

$$round_3 = [0, 1, 2, 8, 6, 5, 3, 4, 7, 9, 10]$$

$$round_4 = [5, 9, 1, 4, 8, 10, 7, 3, 6, 2, 0]$$

$$round_5 = [10, 2, 9, 3, 4, 0, 6, 7, 8, 1, 5]$$

As it can be clearly seen from the example above, during every round, group of cards rotates position within the deck. For example, card 0 moves to the location where 10 was present during round 0(the starting point), card 10 moves to where 5 was and 5 moves to where 0 was. So, 0, 5 and 10 form a rotation group of size 3, such that they return to their original positions after 3 rounds. Similarly, by comparing round$_5$ to round$_0$, we see that 3,4,6,7,8 form a group of 5, and repeat after 5 rounds as they achieve the original state after 5 rounds.

As some of the cards return to original position every 3 rounds, and some every 5 rounds, this can be considered treated as a canonical cycle[1]. Therefore, if array $A$ consists of multiple cycles of varying length, then applying $A$ n times would return to original state, where n is the order of permutation. In case where the number of cycles is different, the least common multiple will return both the chains to the starting state. Due to this inference, the LCM of every rotation group determines the total number of rounds required to return to original state. In the above scenario, one group has a rotation cycle of 3, whereas the other group has a rotation cycle of 5, both will at the original state after $3 * 5$, i.e 15 rotations, which is LCM(3, 5). This is due to the fact that we want all the rotations groups to return to original state, so that the entire deck is in the original state.

## Algorithm

1. Complete one round by following through steps 1 to 3, and obtain the new configuration of the deck.

2. Determine size of all rotation groups between cards in their original configuration versus the cards after one round.

3. Return the lowest common multiple(LCM) between the size of each group.

## Running Time

The running time of the algorithm is $O(n.m)$, where n is the number of cards in the deck, and m is the least common multiple.

# Bibliography

[1] Canonical Cycles, https://en.wikipedia.org/wiki/Permutation#Canonical_cycle_notation_.28a.k.a._standard_form.29