Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

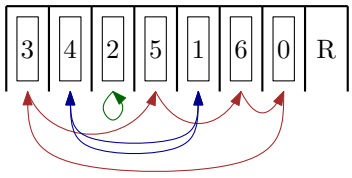
Department of Computer Science
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

3rd November 2016

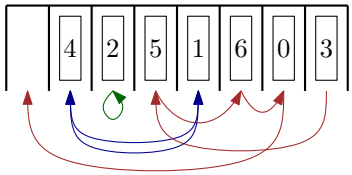
# Data Structures & Algorithm Solutions to Sheet P6 AS 16

**Solution P6.1** *Chinese wall parking.*

For every car, draw an arrow to where it needs to go. For our example permutation, we get the following picture.




If you consider a single cycle of length $l \geq 2$ (for example take the cycle $3 \to 5 \to 6 \to 0 \to 3$ with length 4) we can move any single car to the reserved space (R) to get the following situation.




Now we have broken the cycle and we can move all the cars of the cycle directly to their right positions (in the example in the order 0, 6, 5, 3) using total of $l + 1$ car moves for the cycle.

We do this for every cycle of the permutation independently. Of course we need no moves for cycles of length 1 (cars already in the right position, such as car 2 in the example).

Our program therefore needs to find the lengths of the cycles of the given permutation, discard the cycles of length 1, and print the sum of the remaining lengths increased by 1 (each).

On the lecture website, you can find a solution running in time $O(n)$ that computes the cycle lengths. The solution source contains further comments on the implementation.

### Data

`judge1` $n = 10\,000$, a sorted sequence.

`judge2` $n = 10\,000$, permutation with a single cycle.

`judge3` $n = 10\,000\,000$, random permutation.

**Notes on submitted solutions.** Some of the solutions have used some kind of heuristic to guess the number of moves. For example, let $k$ be the cars in the right positions, and output

$n - k + 1$. This would work only if all the other cars were a part of one cycle. Other solutions tried to improve this by also counting the pairs of cars to be just swapped (like 4 and 1 in the example) and adjusting for them, but that is not enough. Others tried some divisibility conditions or other conditions, but these might work only accidentally.

However, there is a similar approach that actually works: When summing the cycle lengths $l + 1$ in our solution above, we in fact get the number of cars to be moved plus the number of permutation cycles, or $n - k + c$, where $k$ are the cars already in the right spots and $c$ is the number of cycles. However, counting the number of cycles is essentially the same task as counting their length, so the solution algorithm would be almost the same.

The programs that traverse the cycles using recursion crash with a `RUNTIME ERROR` when the cycle length was more than ca 1000, so a non-recursive solution is preferred.

**Mathematical remark.** In a random $n$-element car permutation, the *expected* (average) number of cars in the right position is $k = 1$ (surprisingly), and the expected number of cycles of length at least 2 is $c = \log n - 1$, so the expected number of car moves is $n - 2 + \log n$. See Wikipedia page on random permutations[1] for details.

[1] `https://en.wikipedia.org/wiki/Random_permutation`