Assignment 2 - Algorithms for Sequence Analysis, Summer 2021

Exercise 1: Cyclic permutations (4 Theory)
Given two strings s, t of the same length n, how can you efficiently decide if one is a cyclic permutation of the other? For example, `0123456` and `2345601` are cyclic permutations of each other, but `6543210` is not a cyclic permutation of the other two (it's a reversal).
Give an algorithm that takes the two strings as input and outputs `True` or `False`. Your algorithm should run in $O(n)$ time.

Exercise 2: Bit magic for Hamming distance (4 Theory)
Consider a bit-encoded DNA k-mer for $k \leq 32$ (so it fits into 64 bits). While this problem is in fact independent of the concrete encoding, it may help to have the following default bit encoding in mind: `A` $\mapsto 00$, `C` $\mapsto 01$, `G` $\mapsto 10$, `T` $\mapsto 11$. A k-mer is then encoded by concatenating the bit encodings of the single nucleotides, with leftmost nucleotides getting the most significant bits, i.e., `GCA` $\mapsto (100100)_2 = 36$. Unused bits (if $k < 32$) take the value 0. The *Hamming distance* between two k-mers is the number of positions at which the nucleotides differ.
Write a function that takes two k-mer codes as input and returns `True` if and only if the Hamming distance between the k-mers is ≤ 1. Use only bit operations and avoid loops. Argue why your function is correct.
Hint: After reviewing methods from the lecture, it may help to first think about how to test whether a number has a single 1-bit.

Exercise 3: Patterns with variable-length gap and Hamming distance 1 (4 Theory)
Consider the NFA construction (and its bit-parallel implementation) that allows you to match patterns with variable-length gap, such as the ZNF768 binding pattern

`RCTGTGYRN(17,23)CYTCTCTG.`

(An implementation was provided in a code example for assignment sheet 1; the same implementation is provided again this week.)

Extend the construction in such a way that the automaton additionally matches text substrings with a Hamming distance of 1 to the given pattern. Argue that your construction is correct by stating a lemma about the set of active states after each update step.
Hint: The idea for this exercise is *not* related to the idea for the previous exercise.

Exercise 4: Implementation of Exercise 3 (4 Programming)
Extend the provided code (or re-implement it together with the extension in a language of your choice) to implement the Hamming distance 1 search that you developed in the previous exercise. To access the new feature, add a new option `--allow-mismatch` or `-M`

to the CLI (with help text and all). You may want to read the `argparse` documentation about actions like `'store_true'`.

Remarks

- 50% of points in each category theory and programming (over all exercises and not each assignment sheet separately) are necessary to take the exam.
- You are allowed to work in groups of two and only one of the group members need to submit.
- Submission is via GitHub Classroom (as demonstrated in the lecture).
- Source code must be sufficiently commented and documented to be understandable.
- When using a compiled language, compilation instructions and tools must be provided (e.g., a Makefile).
- In addition to source code, the output must be provided.
- Also, a file AUTHORS with your name(s) must be provided.
- Copying between groups will result in zero points for all involved groups!

Hand in date: Monday, May 03, before 20:00