Algorithms and Data Structures for Biology
18 May 2020

Ugo Dal Lago Guillaume Geoffroy

This assignment will be the third one to be marked. Please carefully read all the instructions below before starting to work on the assignment.

1 Instructions

This assignment must be completed individually, the deadline is on May 27th 2020 at 23.59 CET. To complete the assignment, you need to upload the following file to the course's IOL page (where a specific link will be available):

- One Python file named `FIRSTNAME_LASTNAME.py`, with the code you have developed.

2 The Problem

The problem we want to solve in this assignment is relatively simple to be described, and we are sure this will look somewhat familiar to you. You're given a long strand of DNA, *i.e.*, a string s of n characters in the four-letter alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}$, and you want to know if there are particular *patterns* that are repeated often in s, possibly with small modifications. More precisely, you're given a *pattern length* $m \leq n$ and a *maximum distance* k, and you want to know the string t of length m that is repeated most often in s, with up to k modifications.

For example, if the strand of DNA is $s =$ "GATTACA", the pattern length is $m = 3$ and the maximum distance is $k = 1$, then the best pattern is $t =$ "ATA", has 3 approximate occurrences, at indices 1 ("ATT"), 1 ("TTA") and 4 ("ACA"). Note that, paradoxically, in this case, the best pattern does not appear *exactly* anywhere in the strand.

An exact description of the problem requires some auxiliary mathematical definitions. Given two strings x, y of the same length and in the same alphabet, their *Hamming distance* $H(x, y)$ is the number of positions in which x and y differ (for example, $H(\text{"ATA"}, \text{"ACA"}) = 1$). A *$k$-occurrence* of a string x of length m in a string y of length n is the data of an index $i \leq n - m$ such that $H(x, y[i : m]) \leq k$, where $y[i : m]$ denotes the substring of y that starts at index i and has length m. We denote the number of distinct k-occurrences of x in y by $O_k(x, y)$. Your problem is, given s, m, k as above, to compute the string t which makes $O_k(t, s)$ maximum among all m-character strings.

This assignment asks you to:

1. Write a `Python` program which reads from a file called `inputdata.txt` the following data:
   - a natural number m in decimal notation, in the first line;
   - a natural number k in decimal notation, in the second line;
   - a string s over the alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}$ in the third line;

   and then determines the string t of length m that maximizes $O_k(t, s)$ (or one such string, if there are several possibiities), and `print`s that string.

2. Test it against the example file `inputdata.txt` you can find in the appropriate folder at IOL. If this file is present in the working directory, then running your program in a terminal should produce the following result:

   ```
   your:prompt/$ python3 your-file.py
   AGGGGGC
   ```

   You should create additional test files yourself, and you're encouraged to share the test files with your classmates (but *not* the code).

In particular, we do not ask you to write a proper report this time: the `Python` file will be enough. However, your `Python` program must not use any external module: built-in `Python` features are more than enough to solve this assignment.

Your `Python` program will be judged against the following three parameters, of decreasing importance:

1. **Correctness**: the string t must be one of those maximizing $O_k(t, s)$.
2. **Readability**: your code must be appropriately commented: the structure of it should be understandable by the average `Python` programmer.
3. **Performance**: your program should be as efficient as possible, even when the values of m and k increase.

3 Reading the input file

You can use the following code to read the input file:

```
def readInstance(fileName):
  with open(fileName, "r") as file:
    patternLength = int(file.readline())
    maxDistance = int(file.readline())
    string = file.readline().strip()

    if patternLength < 0: raise ValueError
    if maxDistance < 0: raise ValueError
    for character in string:
      if character not in "ACTG": raise ValueError

    return (patternLength, maxDistance, string)
```