Project 2

Deadline: Sunday, 16 April 2023

Background information

In this project you will work with DNA sequences. Some definitions:

- A DNA sequence S of length n is an n-letter string made up of the letters `"A"`, `"C"`, `"G"`, `"T"`, representing a sequence of nucleotides.
- A k-mer K within S is a k-letter substring of S, i.e. a sequence of k consecutive nucleotides in S.
- The frequency of a k-mer K within S is the number of times the substring K can be found in S.
- A point-x mutation of the k-mer K is another k-mer within S, that differs from K only at the position x, where $x = 0$ corresponds to the first nucleotide in K.

For example, the DNA sequence `"AACGAC"` contains the four 3-mers `"AAC"`, `"ACG"`, `"CGA"` and `"GAC"`, and `"GAC"` is a point-0 mutation of `"AAC"` (and vice-versa).
Throughout this project you may assume that $n \leq 10^8$ and $k \leq 10^5$.

Task 1

Write a Python function that, given a DNA sequence S of length n and numbers $k \in \mathbb{N}$ and `freq` $\in \mathbb{N}$, can find all k-mers within S that occur at least `freq` times. We call these k-mers, if they exist, frequent k-mers in S. In addition, for each frequent k-mer K, your function should also compute the total number of k-mers in S that are point-x mutations of K.
In particular:

- Your function should be defined as `kmer_search(S, k, freq, x)` and return two lists: `L1, L2`.
- `L1` should be a list of lists, and should contain for each frequent k-mer an ordered list of its locations in S. For example, the i-th entry of `L1` is a list of integers, with the first entry being the first location of the i-th found k-mer. Note that this automatically defines each k-mer. The list `L1` itself should be ordered by first occurrence of the k-mers (and not by frequency), so that the first entry of the first entry in `L1` corresponds to the very first location of any frequent k-mer in S.
- `L2`, at position i, should contain the total number of point-x mutations of the k-mer corresponding to the i-th entry in `L1`.
- If there are no frequent k-mers in S, simply return two empty lists.

Your function `kmer_search` may use all available standard types and methods in Python 3, but must work without importing any packages or modules.
Together with your code you should submit a report with a concise description of your algorithm and an explanation of your design decisions for your algorithm. The report should include an analysis of your algorithm's asymptotic complexity. Here you may for simplicity assume that *any* reading or writing access to a `dict`ionary in Python is $\mathcal{O}(k)$, when the key is a string of length k.

Task 2

Single Primer Enrichment Technology (SPET) is a targeted genotyping technology that relies on the sequencing of a region starting at a specific location of a given genome. In a simplified model, this location q has to satisfy the following conditions, given p, k and S:

- q is the starting point of a k-mer K in S.
- the k-mer starts to the left of p, not too far away from p and does not reach p: $p - 2k \leq q < p - k$.
- the k-mer K cannot appear more than 5 times in S,
- the combined proportion of `"C"` and `"G"` within K must be between 35% and 65%: $0.35 \cdot k \leq \#\{c \in K : c = \texttt{"C"} \text{ or } c = \texttt{"G"}\} \leq 0.65 \cdot k$.

Write a Python function that, given a DNA sequence S of length n and numbers $k \in \mathbb{N}$ and $p \in \mathbb{N}$, can find a desired SPET location in S as described above. In particular:

- Your function should be defined as `spet_location(S, k, p)` and return q.
- If there are more than one admissible q, return the one whose k-mer has the lowest frequency in S. Amongst candidates with the same lowest frequency return the value of q that is closest to p.
- If no admissible q can be found, simply return `None`.

Your function `spet_location` may use all available standard types and methods in Python 3, but must work without importing any packages or modules.
Together with your code you should submit a report with a very short description of your algorithm and an analysis of your algorithm's asymptotic complexity. Here you may for simplicity assume that *any* reading or writing access to a `dict`ionary in Python is $\mathcal{O}(k)$, when the key is a string of length k.

Task 3

Test your function `spet_location` on some real-life DNA sequencing data. To this end, download the publicly available DNA sequencing data in FASTA format for the chloroplast of the tomato plant: `https://www.ncbi.nlm.nih.gov/nuccore/NC_007898.3?report=fasta`
You can find more details on the FASTA data format, and on the used notation at the links

- `https://en.wikipedia.org/wiki/FASTA_format`
- `https://en.wikipedia.org/wiki/Nucleic_acid_notation`

Note that before passing the data to your function, it needs to be "cleaned". For example, the FASTA data may contain comments, i.e. lines starting with the character `">"`, and the sequencing data may contain letters other than `"A"`, `"C"`, `"G"`, `"T"`. For the purpose of this task, you should remove any such letters before your analysis. (Your cleaned up file should contain exactly 155461 letters.)
On setting $k = 40$, experimentally find locations p within the tomato DNA data for which your function `spet_location` is able/unable to find a location q. Then implement a Python function `get_my_pqs` that returns a list with three pairs `(p, q)` that you have found experimentally. Here at least one q should be `None`, at least one q should be a number and one p should be in the format `DDMMYY`, where `DD/MM/YY` denotes your birthday. Also state the three values of p, q and the k-mers starting at q in your report.

Extra

If you are interested in the topic, you could also investigate how hard/easy it is to find locations q within other DNA data sets found on the NCBI website. For example:

- the Escherichia coli bacteria:
  `https://www.ncbi.nlm.nih.gov/nuccore/U00096.3?report=fasta`
- the brewer's yeast chromosome 7:
  `https://www.ncbi.nlm.nih.gov/nuccore/BK006941.2?report=fasta`
- the tomato chromosome 6:
  `https://www.ncbi.nlm.nih.gov/nuccore/CM001069.3?report=fasta`
- the human chromosome 21:
  `https://www.ncbi.nlm.nih.gov/nuccore/CM000683.2?report=fasta`

You may share your findings in the report, but this will not influence your pass/fail grade for the project.

Submission via Moodle

Prepare a single Python source file called `project2.py` that contains your functions `kmer_search`, `spet_location` and `get_my_pqs`, together with any routines or classes that you have written and that your functions depend on, but nothing else. Importing your source code via `import project2` should **not** execute any code. The docstring (i.e. the first comment detailing the nature of the module) at the beginning of the file `project2.py` must contain your full name. In addition, prepare a report in PDF format, called `project2.pdf`, that includes your answers for Tasks 1, 2 and 3.