Software Report

Axel Stengel and Romain de Beaucorps

December 2023

1 General structure of the code

The code is divided into 5 python files :

- `report.py` contains `main()` as well as the filtering and output functions and is the file to launch.
- `readfa.py` contains the parsing functions to read and parse `.fa` files. This file was not written by us and taken from https://github.com/lh3/readfq/blob/master/readfq.py.
- `naive.py`, `bfs.py` and `unword.py` contain each associated methods alongside all necessary auxiliary functions.

2 Implementation details

We will only focus on the `unword.py` file since it is the only one with non-trivial implementation.

- The `unword` algorithm uses bit-arrays for space efficiency. However, Python does not have any efficient and native way of creating and handling bit-arrays, especially not using boolean arrays since Python booleans are 24 BYTES long. For this reason, we decided to use the `bitarray.py` module. This module allows us to have a data structure that store one bit in one bit of data. It also gives access to a number of C-coded methods that correspond to our needs.
- For easier handling of the bit array and better code readability, we created a class `q_bit_array` that correspond to the bit array for the words of a given length q. The class has the following attributes :
  - `array`, a bitarray of size 4^q to store the raw data,
  - `len`, an int representing the number of 1 in the bitarray that is modified each time the bitarray is modified. It allows instant access to `array.count(1)`,
  - `full`, a boolean that represent if the bitarray is full of 1s
  - `q`, an int to store the length of the words stored in the structure
  - `max_len`, an int to store the maximum possible number of 1 in the bitarray. It allows instant access to 4^q.

  The class also contains the following methods :
  - `add_word(indexes)` that adds the words in the bitarray and modifies `len` if necessary,
  - `scan(sequences)` that scans the sequences as described in the `unword` section of the project report.

  - `absent_words()` that returns the canonical sequences of the words with a 0 in the bitarray.

- `index_to_word` is the function allowing us to convert a $k$mer encoded into an integer back into its textual form. The implementation is pretty straightforward, apart from two details :
  - Instead of using the expensive exponentiation, we use bitshifts as they are much faster and `x / 4 == x >> 2`.
  - The parameter q : since an A encodes to a 0, we have no way of knowing whether there was a leading A in a $k$mer. This is the reason we pass the length of the original $k$mer, so that even if we are left with a 0 we know to keep decoding to obtain As.