Algorithms for Bioinformatics: Project 1
Definition of Python Class for Sequence Manipulation

Miguel Sozinho Ramalho, UP201403027

March 6^{th}, 2019

1 Introduction

The current report describes a practical approach for creating a Python3 library capable of handling and manipulating information on biological sequences. It is considered paramount that Deoxyribonucleic acid (DNA), Ribonucleic acid (RNA) and Protein (collection of Aminoacids) sequences can be represented and modified. The tool should be able to perform typical operations on those sequences in an efficient manner. Contextualizing, this project belongs to the course of Algorithms in Bioinformatics.

2 Implementation

The main challenge of this project is the architecture design. Namely, how to devise the tool in a way that allows for adding new features, representing the real world structure of the aforementioned sequences while making the most out of Python's design and syntax.

Although an initial approach was conceived with a single class, it quickly became obvious that it was prone to bugs and hardly represented the real world organization. In the end, and following the object-oriented technique studied in classes, an inheritance implementation was designed consisting of a main class **BioSeq** and three direct child classes **DNASeq**, **RNASeq** and **ProteinSeq**.

The mother class contains all the functionality that is applicable to every descendant and each child individually implements the methods that are class specific. Additionally, functions that were abstract and not specific to a class instance were isolated in a **utils** module.

Due to Python's flexibility in handling object attributes it was decided that no **set** methods would be implemented, giving some freedom of usage to the user; it was also decided that all the necessary assertions and checks would be done on the initialization methods of each class. However, the assertion methods should also be accessible by the user to validate any new changes. Also, it was decided that empty sequences are considered valid, otherwise the tool would be too strict and not user friendly.

In order to guarantee a good quality of the tool, it was required to write unit tests for all the developed methods (see Figure 1 for the 100% coverage) and to document every implemented feature.

3 Results

Of the required features, all were implemented, namely:

- Class constructor with sequence and sequence type (**__init__**)
- Methods for visualizing class information (**__str__**, **pretty_print**)
- Sequence validation (**_assert_valid_sequence**, **_assert_seq_type**)

- Symbol/Token frequency (**frequency**)
- GC content (**gc_content**)
- Reverse complement (**reverse_complement**)
- Transcription (**transcription**)
- Translation (**translation**)
- Codon usage (**codon_usage**)
- Open Reading Frames (ORFs) by minimum size (**_reading_frames**, **open_reading_frames**)
- Pretty-printing of some instance information (**pretty_print**)
- Input and Output from/to a file (**save**, **load**)
- Reading FASTA files (**utils.read_fasta**)

Besides the baseline features, some were also implemented, namely:

- Hierarchical representation of classes
- Implementation of special python methods to better handle the sequences (**__str__**, **__repr__**, **__len__**, **__getitem__**, **__getslice__**)
- performance concerns when designing the methods (use of hashmaps, generators, saving as little information on the files as possible, avoid using unnecessary RAM, and others)
- Documentation and testing of all the methods

Coverage report: 100%

Module ↓	*statements*	*missing*	*excluded*	*branches*	*partial*	*coverage*
bioseq\bioseq.py	38	0	0	8	0	100%
bioseq\dnaseq.py	17	0	0	2	0	100%
bioseq\proteinseq.py	10	0	0	0	0	100%
bioseq\rnaseq.py	29	0	0	16	0	100%
bioseq\utils.py	9	0	0	2	0	100%
Total	**103**	**0**	**0**	**28**	**0**	**100%**

Figure 1: Automatically generated test coverage report for the developed Python module

The documentation can be seen by opening the file *docs/_build/html/index.html* in an external browser or by typing **help(bioseq.moduleName)** on the Python console for the desired module.

4 Conclusions

Looking back at all the progress made with this project, the author concludes there was a lot learned and also that most concepts became clearer when they had to be organized so meticulously, as in the current tool. It is his belief that the work produced is of sufficient quality to be used in future projects and also to be extended as more relevant bioinformatics techniques are learned.

5 References

(No paper references required and all other mentions like Python, FASTA, DNA, RNA, Aminoacid, OOP, and others are considered base knwoledge prior to reading this document)