ORF finding

Deliverables:

- findORFs.py - 50 total points
- imports
  - sequenceAnalysis (for FastaReader)
  - your OrfFinder class (placed in sequenceAnalysis)

required output files:

- tass2ORFdata-ATG-100.txt

classes required (minimally):

- OrfFinder
- FastAreader

possible extra credit options - 10 additional points

- minOrf= [integer] This allows exclusion of genes with lengths < minOrf (Note this length includes both start and stop codons)
- biggestOrfOnly True | False This allows reporting of all genes or only the largest within an ORF
- startCodons= ['ATG' | 'TTG' | 'GTG'] This parameter produces a list of start codons used in determining genes (you might convert this to a more useful container)
- stopCodons = ['TAA' | 'TGA' | 'TAG'] This parameter specifies which codons are to be treated as valid stop codons
- note that the given program template is set up for default settings

Due: Monday May 14, 2018 11:55pm

Overview

This lab will give you practice with a real algorithm and with formatted output. We will be finding genes in a genome.

An open reading frame (ORF) is a region of DNA that is free from stop codons (TAG, TAA, TGA). We can then define a putative gene in that ORF by the presence of a start codon at the 5' end of the ORF and a stop codon at the 3' end. Start codons in some bacteria, archaea and viruses can be ATG, GTG or CTG, and in some cases TTG.

Since genes are made up of codons of three symbols, any particular sequence of DNA can be interpreted in each of 3 overlapping coding frames, and since DNA is double stranded, we can have genes located on both strands, yielding 6 possible coding frames for the many genes in a genome. Genes can also overlap each other.

To limit the number of possible genes that we might find, two basic strategies are sometimes used:

- look for genes that are longer than some predefined number (100 bases for example) and
- only consider the largest coding region within an ORF, rather than considering all substrings in the ORF that begin with intermediate start codons.

A well known ORF finder is available at NCBI - ORF Finder https://www.ncbi.nlm.nih.gov/orffinder/ (https://www.ncbi.nlm.nih.gov/orffinder/)

In this assignment, you will write an equivalent tool that provides the same function, though without the graphical display.

Use your sequenceAnalysis module in designing a solution.

Your design is essential for success. Spend time considering how best to solve this problem. Write out pseudocode that describes the core algorithm. This pseudocode is required to be delivered in comments at the top of your code submission. Please start early on this assignment, and consider the algorithm carefully.

- Start and end codons must be in the same frame,
- many possible Met codons (or other possible starts) may exist in an ORF, and
- genes can have significant overlap.
- In some cases, 2 or more ORFS can exist on opposite strands in the same coordinate space.

A partial solution exists in the text using regular expressions (Model p 271). This solution does not handle any of the extra-credit possibilities, and does not handle the required boundary conditions involving gene fragments. Feel free to look, though I did not find this approach helpful. It also uses regular expressions, which are both difficult to read and disallowed for this assignment.

Your implementation should be designed to add capability to your sequenceAnalysis module. Please carefully consider where that capability belongs. You are required to make use of your sequenceAnalysis module, at least for the FastAreader. You might consider adding DNAstring from lab2, or adding a method to calculate reverse complement. You might also add an OrfFinder class to sequenceAnalysis (reccomended) . For this assignment, you are the designer of your toolbox - choose wisely. You might use this again in your final project.

ORF finder

Create a program to analyze a FASTA-formatted file containing a sequence of DNA and find the ORFs (start and stop codons). Your program is called findORFs.py and it:

- reads in FASTA-formatted sequence data from a file called genome.fasta
- finds ORFs that are > 100 nucleotides in length
- writes formatted output data to a text file

- if there are multiple FASTA-formatted sequences in the file, you should perform Orf-finding on each of them independently.

Design specification

Input file

I will provide 2 test files for your use. For debugging purposes, use lab5test.fa, which is provided in Canvas::files. As your final test, use the tass2.fa genome. I have provided the output generated by my program for your comparison. The tass2.fa genome should be used to generate: tass2ORFdata-ATG-100.txt.

Output file

Write your tass2 output to a file named: tass2ORFdata-ATG-100.txt. This file should be generated with the following program options:

- The minimum gene size to report is 100 nucleotides (including stop and start)
- The only start codon to consider is ATG
- Only the largest putative gene in an ORF is reported
- The input file is tass2.fa

I have provided the output generated using a minimum gene size of 300, which you can use for testing.

Gene finding

Your program should find genes in each of the six possible coding frames. The genes that you find will begin with a start codon and end with a stop codon, with the exception of those gene-fragments that might be located on the ends of the sequence (see boundary conditions).

Boundary conditions

You may find that an ORF exists at either or both termini of the sequence you are examining. This may mean that you will only be seeing a gene fragment located at the sequence termini.

For example, consider the sequence:
AAA AAA AAA TGA CCC CCC ...

Here, we see a stop codon with a set of AAA codons located upstream. In this case, assume that a start codon exists upstream of our sequence, and report this as a gene in Frame 1, starting at position 1, ending at position 12 and having a length of 12.

The same may occur on the 3' end of the sequence, for example:
A AAA AAA ATG CCC CCC CCC CC

In this case, report a gene in Frame 2 starting at position 8, ending at position 21 and having a length of 14.

And, of course genes can exist on the opposite strand, so:
TTA AAA AAA AAA CAT CC
would be reported in Frame -3, starting in position 1, ending at position 15, with length 15.

NCBI ORF finder is a good test case to see what your output should look like. THe NCBI program does not handle this boundary case however.

Sort Order

Your output should be sorted by decreasing ORF size, and in cases where multiple ORFs have the same size, sort those by the left position of the gene. (build a key that makes this easy).

Minimum gene size (option)

The minimum gene size to report is 100 nucleotides. This count of nucleotides includes the start codon and stop codon.

Largest ORF (option)

In any ORF, there may be multiple start codons present. For the default assignment, only consider the largest gene in an ORF. For the extra-credit assignment, if this option is not set, report all ORFs found (subject to the minimum size option)

Start codon(s) (option)

For the default part of the program, only consider ATG as a start codon. For extra credit, use an input parameter that specifies a set of start codons.

Stop codon(s) (option)

For the default part of the program, only consider TAG, TAA and TGA as stop codons. For extra credit, use an input parameter that specifies the set of stop codons.

Report output

All positions reported use the coordinate position of the top strand, and the first position in the sequence (on the left) is position 1. This coordinate system is used for all 6 coding frames.

Frames 1 through 3 can be determined using the start position of the gene. Assuming the position is p, (p mod 3) + 1 will provide the coding frame. For genes on the bottom strand, the frame is based on the right hand side of the sequence (see below).

For each putative gene, your report must provide:

- coding frame. for the top strand, a gene starting in the first position in the sequence is in frame 1.

For the bottom strand, frames are numbered based on the right-hand coordinate position of the sequence. So, if the sequence is 100 bases long, and you have a gene in positions 89..100, that gene is in frame -1. You can still use the coordinate positions, but the calculation is: - ((len(sequence) - last position) mod 3 + 1). Remember that the last position in this case corresponds to the start codon of the gene on the bottom strand. These conventions make sense if you think about the reverse complement of the original sequence. The coding frame should be reported with a + or - as needed to describe the coding frame

- start position. This position should correspond to the first base of the start codon (top strand) or last base of the stop codon (bottom strand), or the terminus of the genome if this is a gene fragment.
- end position. This position should correspond to the first base of the start codon (bottom strand) or last base of the stop codon (top strand), or the terminus of the genome if this is a gene fragment.
- length. This includes the start and end codon

Line 1 - the header line from the original sequence (without >)

Lines 2 .... (use '{:+d} {:>5d}..{:>5d} {:>5d}' or equivalent)

+1 57166..61908 4743
...

Extra Credit - 10 points possible

- Allow your program to optionally use a set of start codons - ATG, GTG, TTG for example.
- Allow your program to optionally use a set of stop codons - TAG, TGA, TAA for example.
- Allow your program to print every putative gene in an ORF instead of only the largest.
- Allow your program to have varying ORF size minimums

Submit your code and answers

For this lab, you should upload the following three files as attachments:

- your sequenceAnalysis.py module
- findORFs.py, include design in # comments at the beginning of findORFs.py
- tass2ORFdata-ATG-100.txt

Important: to get full credit on this lab assignment, each of the code files you submit code needs to:

- Run properly (execute and produce the correct output)
- Include an overview about what your program is designed to do with expected inputs and outputs
- Include design (place at the top of your file below your name, group, and description). Include any assumptions or design decisions you made in writing your code
- Adhere to the Report format specification
- Include docstrings for Program, Module, Class, Methods and Functions
- Contain in-line comments

Congratulations, you finished your fifth lab assignment!

Hints:

One strand at a time

One design solution considers only the top strand, then generate the reverse complement and a second list of gene candidates. When done in this way, remember to fix the start/end coordinates that you get from the reverse complement solution, since those will be based on the other end of the sequence.

An alternative design exists for the reverse strand that does not require generation of the reverse complement. This solution is interesting and may be simpler to consider. This solution scans the bottom strand left to right, finding stop codons first (reverse complement), then looking for start codons (reverse complement).

A third solution for the bottom strand would scan right to left, finding starts until a stop is encountered.

Start and Stop handling

If you are going to do the extra-credit, then you will need to remember where the starts are until you find a stop. This is straightforward if you place those positions in a list, organized by frame. Even if you are not doing the extra-credit, placing starts on a list organized by frame will really simplify your code. This solution would require three lists for each of the three reading frames.

As an alternative, you could scan each of the three reading frames independently, counting by threes.

When you do find a stop, you can then consider the start(s) that you found previously. Notice that the "longest" ORF is then defined by the start that is at the beginning of your start list. Don't forget to clean up this list every time you find a stop that would terminate all of those "pending" genes. Notice what would happen if you didn't clean up your start list - you would be finding ORF candidates that had a stop in the middle of them.

Termini and gene fragments

This is a bit easier if you think about a start to your list for each of the 3 coding frames. I used negative positions. Notice that if there happens to be a start at the first position of the sequence, then frame1 is already cared for.

Path for your input files

All of the input files and programs can be in the same folder so you won’t have to specify a path, if your data file is in some other folder, you can provide that like this:
~/Desktop/bme160/someOtherPlace/labData.fa.

Comments

To get full credit for this lab, you must turn in your design for each program you submit. Remember that any item you write or include must have proper docstrings.

In [5]:

```
#!/usr/bin/env python3
# Name: Your full name (CATS account username)
# Group Members: List full names (CATS usernames) or “None”


##################################################################################
# CommandLine
##################################################################################
class CommandLine() :
    '''
    Handle the command line, usage and help requests.

    CommandLine uses argparse, now standard in 2.7 and beyond.
    it implements a standard command line argument parser with various argument
options,
    a standard usage and help.

    attributes:
    all arguments received from the commandline using .add_argument will be
    avalable within the .args attribute of object instantiated from CommandLine.
    For example, if myCommandLine is an object of the class, and requiredbool wa
s
    set as an option using add_argument, then myCommandLine.args.requiredbool wi
ll
    name that option.

    '''

    def __init__(self, inOpts=None) :
        '''
        Implement a parser to interpret the command line argv string using argpa
rse.
        '''

        import argparse
        self.parser = argparse.ArgumentParser(description = 'Program prolog - a
brief description of what this thing does',
                                              epilog = 'Program epilog - some oth
er stuff you feel compelled to say',
                                              add_help = True, #default is True
                                              prefix_chars = '-',
                                              usage = '%(prog)s [options] -option
1[default] <input >output'
                                              )
        self.parser.add_argument('inFile', action = 'store', help='input file na
me')
```

```
        self.parser.add_argument('outFile', action = 'store', help='output file
name')
        self.parser.add_argument('-lG', '--longestGene', action = 'store', nargs
='?', const=True, default=False, help='longest Gene in an ORF')
        self.parser.add_argument('-mG', '--minGene', type=int, choices= (100,200
,300,500,1000), default=100, action = 'store', help='minimum Gene length')
        self.parser.add_argument('-s', '--start', action = 'append', default = [
'ATG'],nargs='?', help='start Codon') #allows multiple list options
        self.parser.add_argument('-t', '--stop', action = 'append', default = ['
TAG','TGA','TAA'],nargs='?', help='stop Codon') #allows multiple list options
        self.parser.add_argument('-v', '--version', action='version', version='%
(prog)s 0.1')
        if inOpts is None :
            self.args = self.parser.parse_args()
        else :
            self.args = self.parser.parse_args(inOpts)

#############################################################################
# Main
# Here is the main program
#
#
#############################################################################


def main(inCL=None):
    '''
    Find some genes.
    '''
    if inCL is None:
        myCommandLine = CommandLine()
    else :
        myCommandLine = CommandLine(inCL)
    print (myCommandLine.args)
###### replace the code between comments.
        # myCommandLine.args.inFile has the input file name
        # myCommandLine.args.outFile has the output file name
        # myCommandLine.args.longestGene is True if only the longest Gene is des
ired
        # myCommandLine.args.start is a list of start codons
        # myCommandLine.args.minGene is the minimum Gene length to include
        #
#######

if __name__ == "__main__":
    main([  'tass2.fa',
            'tass2ORFdata-ATG-100.txt',
            '--longestGene'])  # delete the list when you want to run normally
```

```
Namespace(inFile='tass2.fa', longestGene=True, minGene=100, outFile=
'tass2ORFdata-ATG-100.txt', start=['ATG'], stop=['TAG', 'TGA', 'TAA'
])
```

findORFs output using tass2.fa at minOrf=300 cutoff

In [ ]:

```
tass2 NODE_159_length_75728_cov_97.549133
+1 57166..61908  4743
-1  8192..11422  3231
+2 65963..69004  3042
-3 14589..16862  2274
-2  2968.. 4872  1905
+1 64093..65952  1860
-3    30.. 1694  1665
+1 69475..71052  1578
+1 48805..50223  1419
+1 47398..48798  1401
-3 29133..30500  1368
-1 40922..42250  1329
-2 19975..21270  1296
+3 72273..73559  1287
-1 24482..25639  1158
+1 50251..51366  1116
-1  6689.. 7804  1116
-3 27501..28601  1101
+2 63038..64078  1041
+3 62019..63038  1020
-2 42271..43263   993
+3 51864..52805   942
+1 45484..46371   888
-3 11433..12317   885
+2 74357..75184   828
+2 71576..72394   819
+3 46341..47138   798
-2 18613..19407   795
+1 55642..56388   747
-1 16940..17632   693
-3 26115..26801   687
-2 13288..13974   687
-1 30998..31654   657
-1 21338..21994   657
-2 12601..13251   651
-2  4894.. 5532   639
-3 32592..33221   630
+1 53977..54588   612
-1 39914..40525   612
+3 54588..55193   606
-3 33234..33809   576
-3 22002..22559   558
-3 23859..24413   555
-2  1945.. 2490   546
+1 73861..74370   510
+1 16324..16806   483
```

```
-2  6214.. 6696   483

-1 22556..23032   477
+2 53324..53776   453
+3 32808..33260   453
+1 29056..29502   447
-2 36286..36729   444
+3 51396..51833   438
+2 55196..55627   432
-1  2468.. 2896   429
-2 31798..32220   423
+1 52891..53307   417
-2 30595..30996   402
-2  5809.. 6204   396
+1     1..  393   393
-2 35740..36129   390
+2 56474..56857   384
-3 34542..34925   384
-1 17888..18268   381
-3 23004..23381   378
-3 52968..53336   369
-1 32207..32572   366
+2 36419..36775   357
-1 23396..23752   357
+2 44594..44935   342
-2 18268..18609   342
+1 43714..44049   336
+2  6557.. 6886   330
-2 34927..35253   327
-3 34131..34451   321
+1 30778..31095   318
+2 24974..25288   315
+1  1246.. 1557   312
-2 39160..39468   309
-3 35253..35561   309
-2 68932..69234   303
```

findORFs output using lab5test.fa at minORF=0 cutoff

In [ ]:

```
test
+1      1..     9      9
+2      1..     4      4
test2
+2      1..    10     10
+3      1..     5      5
test3
+3      1..    11     11
+1      1..     6      6
test-1
-1      1..     9      9
-2      6..     9      4
test-2
-2      1..    10     10
-3      6..    10      5
test-3
-3      1..    11     11
-1      6..    11      6
test1A
+1      1..     9      9
+2      1..     4      4
test2A
+2      1..    10     10
+3      1..     5      5
test3A
+3      1..    11     11
+1      1..     6      6
test-1A
-1      2..    10      9
-2      7..    10      4
test-2A
-2      3..    12     10
-3      8..    12      5
test-3A
+2      1..    13     13
-3      3..    13     11
-1      8..    13      6
```