CAB201 Assignment, semester 2, 2019.

Genomic Sequence Retrieval

NCBI: "The National Center for Biotechnology Information is part of the United States National Library of Medicine, a branch of the National Institutes of Health. The NCBI is located in Bethesda…" (follow the link to learn more background information)

In this assignment we are interested in a particular DNA repository, commonly referred to as 16S. The Wikipedia article 16S ribosomal RNA describes some of the background information. A collection of curated 16S DNA sequences from NCBI will be used in this assignment. The assignment is concerned with managing access to the collection. The biological science is out of context, but we are concerned with writing a C# program to facilitate command line access to the data file.

A large file (the 16S collection) is made available to you. The file format is described later in this document. You task is to provide a command-line interface that supports search access to the collection. The assignment consists of two deliverable parts. Part I is concerned with simple indexing, searching, and displaying of the information, using exact match searches. Part II is concerned with more complex search tasks, focusing on inexact (partial match) search.

The 16S collection.

A file named 16S.fasta.zip is available to you on BB, in the assignment folder. Download and unzip the **16S.fasta** file into your working directory. Make sure that you keep all the data and programs on your own disk storage (USB disk or laptop) because laboratory computers will not keep your data. Keep the data on the server, and a copy on your own storage device or PC.

The **FASTA file format** is simple. There are 20,767 genomic (DNA) sequences in the file. Each sequence is represented by two consecutive lines. The first line begins with the character '>' and contains meta-data describing the source of the DNA sequence, its identity, and other details. The following line contains the respective DNA. For instance, the following 4 lines of text correspond to two sequences; note that the DNA lines are truncated:

>NR_118941.1 Streptococcus alactolyticus strain NCDO 1091 16S ribosomal RNA, partial sequence
GAACGGGTGAGTAACGCGTAGGTAACCTGCCTTGTAGCGGGGGATAACTATTGGAAACG
ATAGCTAATACCGCATAACAG…
>NR_115365.1 Streptomyces albus strain CSSP327 16S ribosomal RNA, partial sequence
TCACGGAGAGTTTGATCCTGGCTCAGGACGAACGCTGGCGGCGTGCTTAACACATGCAA
GTCGAACGATGAAGC…

The metadata line always starts with a '>' character. The first sequence-id is **NR_118941.1**, the name of the organism is Streptococcus alactolyticus, and there is some more descriptive information about the sequence. The next line is the DNA representation, extracted from the 16S region of the bacterial RNA.
The second sequence-id is **NR_115365.1**, the name is Streptomyces albus, and more meta-data follows. That line is followed by another line with the DNA representation.
Note that the length of the DNA sequence can vary; it is not fixed.

There is one complication with the file format that you must be aware of, and overcome in solving the access requirements. There are 306 sequences in the file where the metadata lists more than one source, but the same DNA. For instance, the following is an example of an identical DNA sequence which was recorded 3 times (from independent sources):

>NR_074334.1 Archaeoglobus fulgidus DSM 4304 16S ribosomal RNA, complete sequence
>NR_118873.1 Archaeoglobus fulgidus DSM 4304 strain VC-16 16S ribosomal RNA, complete sequence >NR_119237.1 Archaeoglobus fulgidus DSM 4304 strain VC-16 16S ribosomal RNA, complete sequence
ATTCTGGTTGATCCTGCCAGAGGCCGCTGCTATCCGGCTGGGACTAAGCCATGCGAGTCA
AGGGGCTTGTATCCCTTCGGGGATGCAAGCACCGGCGGACGGCTCAGTAACACGTGGAC
AACCTGCCCTCGGGTGGGGGATAACCCCGGGAAACTGGGGCTAATCCCCCATAGG…

When considering your solutions to the access requirements, bear this in mind, lest your search miss some sequence metadata.

Access requirements

Users need to be able to run queries against the collection. These queries vary in difficulty from rudimentary to complex. In this assignment you are required to facilitate access to the 16S sequences file, supporting various ***search functionality,*** supporting several complexity ***levels*** as described below. An important mandatory requirement is that the 16S.fasta file must not be loaded into memory – the search is disk based. This is because in the intended applications the file size may be too large to fit in memory, or some other resource constraints prevent the application from having access to enough memory – e.g. in a shared environment.

These are the access requirements:

1. Sequential access using a starting position in the file.

Example: ***Search16s -level1 filename.fasta 273 10***

Search16s is the program name, ***-level1*** is a flag indicating the type of search required (in this case, level 1 is for finding a sequence by ordinal position), ***filename.fasta*** is the fasta file name to read from, ***273*** is the **line number** to start the output from, and ***10*** is the number of **sequences** that the program should output. Remember that a sequence consists of two lines, and therefore the **line number** that the user enters must be an odd number, and within the range of lines in the file.

Program output is displayed to the console.

Given a starting line, list the required number of sequences to the screen, starting from the given line number, and showing both lines that correspond to the sequence, i.e. the meta-data line and the DNA line.
All the output goes to the console. Since there are **two lines per sequence** (metadata line and DNA line), the above program execution will result in the output of 20 lines from the input file, corresponding to 10 sequences.

Note that sequence line number in the file carries no information other than the ordinal position of the sequence in the 16S file. For instance, Sequence 11 appears on lines 21 and 22 in the 16S.fasta file and sequence 95 appears on lines 189 and 190 in the 16S.fasta file.

The program output must be **identical** to the text in the input file. This is important since the program output will be compared to the expected output, and every character matters – even spaces. No additional formatting is allowed. If the program is called and asked to start from line 1, and display all the sequences, then the output must be identical to the input file (i.e. the file is essentially copied from the input file to the console.)

Example call: ***Search16s -level1 16S.fasta 273 1***

Program output:

>NR_025900.1 Thermus aquaticus strain YT-1 16S ribosomal RNA, partial sequence
GCTCAGGGTGAACGCTGGCGGCGTGCCTAAGACATGCAAGTCGTGCGGGCCGTGG
GGTATCTCACGGTCAGCGGCGGACGGGTGAGTAACGCGTGGGTGACCTACCCGGA
AGAGGGGGACAACATGGGGAAACCCAGGCTAATCCCCCATGTGGACACAT…

You can verify that your program works correctly by running it over a smaller file and comparing the input file with the program output, which you can redirect to a file, from the command line, when you run the program.

Example:

Search16s -level1 16S.fasta 273 N > myfile.fasta

The program will capture the output in a file called ***myfile.fasta*** in the current directory. N is the number of sequences to output (not the number of lines, which is 2N)

A suitable error message should be displayed if the input cannot be acted upon, for whatever reason. Think about possible user input errors and write code to respond to it with meaningful error messages.

2. Sequential access to a specific sequence by sequence-id.

Example: ***Search16s -level2 16S.fasta NR_115365.1***

The program will respond by displaying to the console the respective sequence lines, or a suitable error message if not found.

Program output could be something like this:

>NR_115365.1 Streptomyces albus strain CSSP327 16S ribosomal RNA, partial sequence

TCACGGAGAGTTTGATCCTGGCTCAGGACGAACGCTGGCGGCGTGCTTAACACATG
CAAGTCGAACGATGAAGCCCTTCG…

If the sequence-id cannot be found then output an error message
For instance: *Error, sequence NR_999999.9 not found.*

3. **Sequential access to find a set of sequence-ids given in a query file, and writing the output to a specified result file.**

Example: ***Search16s -level3 16S.fasta query.txt results.txt***

Query filename – in the example it is **query.txt** – is specified on the command line. The file contains a set of sequence-ids. The program writes out all matching sequences to the specified results file (not to the console). Sequence-ids that are not found should however be output to the console with a suitable error message.

Suppose that the input file ***query.txt*** contains 3 lines:

NR_115365.1
NR_999999.9
NR_118941.1

Further suppose that the second sequence-id cannot be found.
The output file, **results.txt,** will contain ONLY the sequences that were found, first and third in the list above, with the same format as with previous levels.

For each sequence-id that is **not found** a message should appear **on the console** (it should not appear in the output file!!).
So in the example above, a message like this should appear on the console:

Error, sequence NR_999999.9 not found.

Display suitable error messages if the program call contains any errors.

Important – remember that some DNA sequences have more than one sequence ID.

4. Indexed file access, implementing direct access to sequences.

1) **Before running any searches, the fasta file must be indexed.**

   Example: ***IndexSequence16s 16S.fasta 16S.index***

   Two file names are specified, the fasta file name and the index file name (these are run-time variables, not fixed hard-code file names). The program creates a *sequence-id index* to the fasta file. The index supports direct access to sequences, by sequence-id. Specifically, the program creates an index file. Each line consists of a ***sequence-id*** and ***file-offset***. **Note that the index files indexes sequences, not lines**.

   The indexing program, ***IndexSequence16s***, should be implemented as a separate Visual studio project.

   An example index file:

   NR_115365.1 0
   NR_999999.9 531
   …
   NR_118941.1 1236733
   …

   The first sequence above if found on a line with byte offset 0 from the beginning of the file, the second sequence can be found at offset 531 from the beginning of the file, and the next sequence above (somewhere, later in the file) can be found at offset 1236733 bytes.
   The specific details of programming this can be found in lecture notes and in chapter 14 of the textbook, as well as in many online forums. You essentially need to use the FileStream class; the ***position*** property gives the file offset, while the ***seek()*** method allows direct access to a byte offset in the file.

2) The index file is created once only, and afterwards the file can be searched many times using the index file to facilitate direct access to the sequences file.

   Example: ***Search16s -level4 16S.fasta 16S.index query.txt results.txt***

   The program operates in the same manner as the level 3 program, but instead of using a sequential file scan, it uses the index, specified with the additional index filename. When the program starts it loads the index file into memory. The index file is much smaller of course, and we assume that it fits in memory. The program then uses it to search for query sequences-ids. The sequences are then read from the fasta file using ***direct access*** rather than a sequential scan by using the file offsets from the index, and the ***Seek()*** method to go directly to the sequences on disk.

5. Search for an exact match of a DNA query string.

Example:

Search16s -level5 16S.fasta CTGGTACGGTCAACTTGCTCTAAG

The solution is based on a sequential file scan.
Display all matching sequence-ids to the console.

For instance:

NR_115365.1
NR_123456.1
NR_118941.1
NR_432567.1
NR_118900.1

Display suitable error messages if the program call contains any errors or if no matching sequences can be found.

6. Search for a sequence meta-data containing a given word

Example:

Search16s -level6 16S.fasta Streptomyces

The solution must be based on a sequential scan.
Display matching sequence-ids to the console. A matching sequence must contain the query word in the meta-data. For instance, for the above query you may display to the console:

NR_026530.1
NR_026529.1
NR_119348.1

These sequence-ids correspond to sequences that match the word ***Streptomyces***

>NR_026530.1 **Streptomyces** macrosporus strain A1201 16S ribosomal RNA, partial sequence
GACGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAACGATGAACCTCCTTCGGGAG
GGGATTAGTGGCGAACGGGTG…
>NR_026529.1 **Streptomyces** thermolineatus strain A1484 16S ribosomal RNA, partial sequence
GACGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAACGGTGAAGCCCTTCGGGGTG
GATCAGTGGCGAACGGGTGAG…
>NR_119348.1 **Streptomyces** gougerotii strain DSM 40324 16S ribosomal RNA, partial sequence
AACGCTGGCGGCGTGCTTAACACATGCAAGTCGAACGATGAAGCCCTTCGGGGTGGATT
AGTGGCGAACGGGTGAGTAAC…

Display suitable error messages if the program call contains any errors.

7. Search for a sequence containing wild cards.

A "*" stands for ***any number*** of characters in the same position.
Display all matching sequences to the screen

Example: ***Search16s -level7 16S.fasta ACTG*GTAC*CA***

This should match, for instance

ACTGGTACCA
ACTGCGTACCA
ACTGGTACGCA
ACTGAGTACTCA
ACTGACGTACTGTGCCA
ACTGACCGTACTGCA
ACTGGTACTGTCA
etc.

Hint: you can use a regular expression.
Self-study - this is not covered in lectures.
This level is optional.

General Assignment Requirements

The assignment consists of **2 deliverable parts**, **with different due dates.**

Part I – worth 20% of subject grade, levels 1-3 only, marked out of 100.

Progress submission.	Due date: week 7, September 8th, 2019, 22:00
Final submission levels 1-3.	Due date: week 8, September 15th, 2019, 22:00

Part II – worth 30% of subject grade, levels 4-7, marked out of 100

Part II must include all part I functionality – it is an extension of the same Visual Studio project from Part I.

Progress submission.	Due date: week 11, October 13th, 2019, 22:00
Final submission levels 4-7.	Due date: week 12, October 20th, 2019, 22:00

It is possible to get 100 marks for implementing parts 4-6 only.
Level 7 is optional, for additional 10/100 marks.
However, the maximum score for part II is still only 100 marks.
The optional marks can only offset for marks lost in marking levels 4-6.

You are required to submit a progress report for each part, one week before the final due date. A progress report consists of a zipped file with your visual studio project, consisting of

a) **ALL** work completed to-date – even if it is incomplete
b) a statement of completeness – describing what you have achieved so far.

The progress report is mandatory, and it is worth 5/100 marks for each assignment part - but the marks will only be awarded if a final submission of an assignment part is made by the due date.

The final submission of a program part (I, and II) will consist of:

1. The zipped Visual Studio directory
2. Statement of completeness
3. Program user manual, describing how to use it.
4. A program test report.
5. You will also have to fill in a **self-assessment CRA sheet** where you will rate your submission according to the criteria.

More detailed submission instructions will follow in due course.

Performance Requirements:

A detailed CRA will provide information about the marks allocated to various aspects of this assignment.

Part-I, levels 1-3 will be assessed out of 100 marks, subject to the CRA being fully met.
Part-II, levels 4-6 will be assessed out of 100 marks, subject to the CRA being fully met.
(With levels 1-6 sufficient for 100 marks; level 7 is worth up to 10 bonus marks, but not exceeding 100 marks for the assignment.)

NOTE: **part II must include Part I**. Submit only one project version supporting all the functionality that you were able to implement, and document it in your statement of completion. For instance, you may have only completed levels 1,2,3,5,6. Your Part II solution must support all the functionality you declare as completed.

Some important aspects of assessment to bear in mind from the outset are

1) Program correctness
2) OOP program design
3) Code efficiency
4) Code documentation
5) User manual – detailed instructions on program use
6) Test report – a report, in PDF format, describing the various tests that you performed, and documenting the program run (screen shots.)
7) Statement of completeness – an overall summary, identifying which levels were implemented, and it **must list all known limitations, deficiencies and/or bugs**. Failure to identify bugs or deficiencies will lead to separate deductions. So marks can be lost for faulty program, as well as for not detecting and declaring the faults. If you declare faults you will lose marks only for the faults, but not for not detecting and reporting it.

Finally – this is an individual assignment.
Make yourself familiar with QUT rules regarding
Academic honesty and plagiarism