CAB201 Programming Principles - Semester 2, 2019 Report for Assignment: Project – Genomic Sequence Retrieval - Part II

Student name and number:

[Phuong Nam Ly n10098097]

The testing report is included separately.

Build and Run Instructions

The following instructions have been tested:

1. Place your project (exactly what you are about to upload as a submission) on a USB disk
2. Log onto a QUT SEF computer lab – this must be a **new login session**
3. Follow the program user manual
4. Test your program.

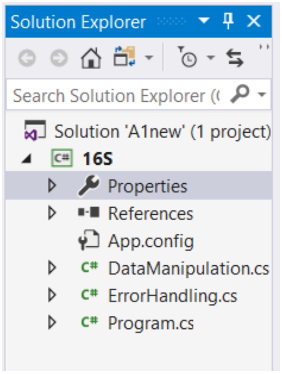
Program User Manual

Step-by-step instructions:

Decompress the A1new.zip file

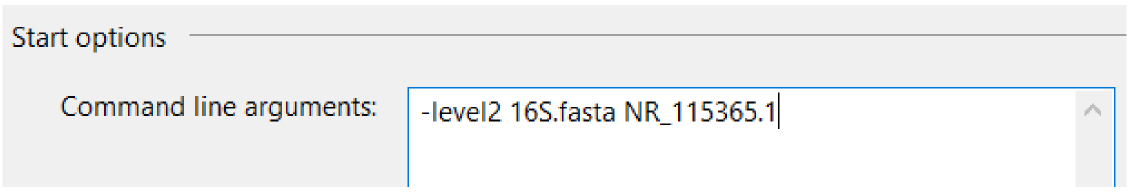
Open the solution file A1new.sln, which is located inside of the decompressed A1 folder

Open the Properties, which is in the Solution Explorer



In the "Debug" section, enter the command line.

A sample command line is entered.

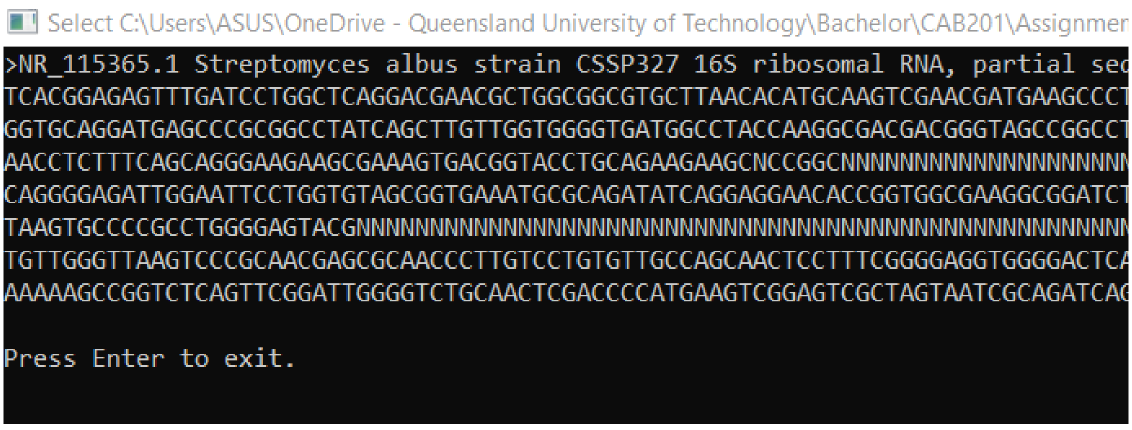


Start options

Command line arguments: -level2 16S.fasta NR_115365.1

Start the program in Debug mode by clicking the Start button (green) or use Ctrl + F5.

Wait for the output.



Select C:\Users\ASUS\OneDrive - Queensland University of Technology\Bachelor\CAB201\Assignmen

```
>NR_115365.1 Streptomyces albus strain CSSP327 16S ribosomal RNA, partial sec
TCACGGAGAGTTTGATCCTGGCTCAGGACGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAACGATGAAGCCCT
GGTGCAGGATGAGCCCGCGGCCTATCAGCTTGTTGGTGGGGTGATGGCCTACCAAGGCGACGACGGGTAGCCGGCCT
AACCTCTTTCAGCAGGGAAGAAGCGAAAGTGACGGTACCTGCAGAAGAAGCNCCGGCNNNNNNNNNNNNNNNNNNNN
CAGGGGAGATTGGAATTCCTGGTGTAGCGGTGAAATGCGCAGATATCAGGAGGAACACCGGTGGCGAAGGCGGATCT
TAAGTGCCCCGCCTGGGGAGTACGNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
TGTTGGGTTAAGTCCCGCAACGAGCGCAACCCTTGTCCTGTGTTGCCAGCAACTCCTTTCGGGGAGGTGGGGACTCA
AAAAAGCCGGTCTCAGTTCGGATTGGGGTCTGCAACTCGACCCCATGAAGTCGGAGTCGCTAGTAATCGCAGATCAG

Press Enter to exit.
```

The program is applicable for Build and Run Instructions

Statement of Completeness

This statement of completeness will need to *accurately* state the functionality which has been implemented. There will be a penalty of 3.5 marks (loss of 3.5 marks) for a non-completed or submitted statement of completeness, and a penalty of 1 mark for each inaccurate statement to a maximum of 3 marks.

In the following section, you are required to mark which functionality you have implemented. <u>In the column on the right please mark 'Y' where you have completed this functionality, and 'N' where you have not</u>. Please fill in any additional text boxes requested, and please note any limitations or bugs in the box at the end of each section. You may expand the table if you need more room for comments.

Basic Functionality		
Build & Run	When following the Build & Run instructions, the program successfully builds and runs. This was tested in a QUT SEF lab with a new login session, using the same zipped folder that is submitted.	**Y**/N
Basic itinerary output	The program displays the data from the file	**Y**/N
	The program displays the appropriate line	**Y**/N
	The correct amount of information is displayed, e.g. only the relevant entries	**Y**/N
	The correct level, provided as a command line flag ***-levelN***, is executed	**Y**/N
	The program **does not** store the whole file in memory, instead it accesses the file on disk	**Y**/N
Error handling	A clear error message is displayed when an incorrect number of arguments is provided	**Y**/N
	A clear error message is provided when an incorrect flag is provided (e.g. not -level1, etc.)	**Y**/N
	A clear error message is provided when the input file doesn't exist, or is incorrectly formatted	**Y**/N
Comments	For basic functionality, I think there are no limitations, bugs logical errors for Level 4-7 as my testing reports show there are no difference between the expected and the actual results. I have done the test on a QUT computer.	

Searching Algorithm – Part II Please underline, circle or highlight the levels that were completed.	
Algorithm	**Level:** Level 4, level 5, level 6, level 7
	Bonus: Sequence matching using wildcards
Comments	I think there are no limitations, bugs logical errors for Level 4-6 as my testing reports show there are no difference between the expected and the actual results. However, my testing file 16S_test.fasta is small compared to the 16S.fasta so I assume there would be bugs.

Level 7 Explanation:

Theoretical approach:

The symbol '*' from the wild card sequence immediately reminds me of '>', which I used to split the metadata line a lot in the assignment. If I split the wild card sequence with '*', it will give me a list of substrings of the wildcard sequence.

It is obvious that a DNA line must contain all the substrings. However, what are the other possible conditions?

Next, I draw some graphs and I figure **that the DNA line needs to contain the substrings in a sequential order.** This means:

```
If ((The piece that contain substring A + some string + The piece that contain substring B + The rest of the DNA line) Contains (Substring A))
{
    If ((some string + The piece that contain substring B + The rest of the DNA line) Contains (Substring B))
    {
        Go on until the last substring is found or until a search fails
    }
}
```

Coding approach:

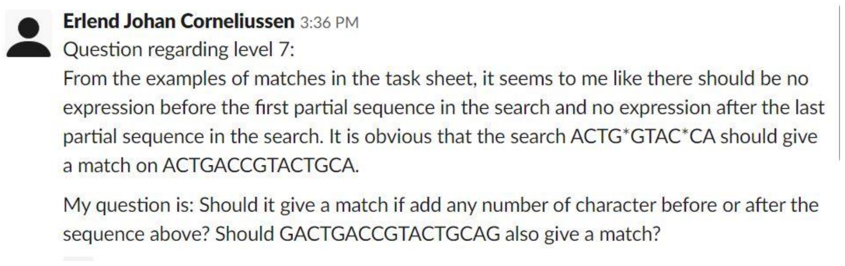
I use Google to find what is the equivalent of "**the DNA line needs to contain the substrings in a sequential order**" in C#. The closest thing I get is this.

https://stackoverflow.com/questions/2641326/finding-all-positions-of-substring-in-a-larger-string-in-c-sharp

At first, I thought I need to find all occurrences of substring A, then check if substring B and so on, follows. After a while, I realize that I do not need to find all occurrences of the substring A because only the first substring A, the first substring B, and so on, matter as long as the search ignores everything prior to the first substring A, the first substring B, and so on. By using indexOf() and the increment index += substring.Length, inside a for loop, I do not have to remove everything prior to a particular substring that I'm searching, as for loop only search for the index after the previous searching.

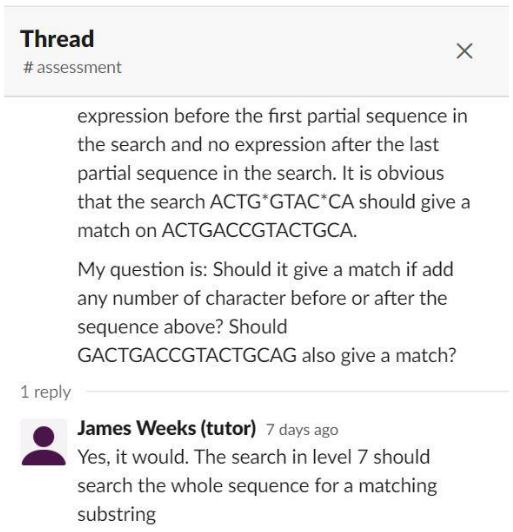
In the for loop, if the search fails, index is 0 and the program breaks out of the loop. If the index search for the last substrings within the wild card sequence is successful, the line satisfies my requirement: **the DNA line needs to contain the substrings of the wild card sequence in a sequential order** so the program outputs the corresponding id to the console.

Checking Slack to for further verifications:



Erlend Johan Corneliussen 3:36 PM
Question regarding level 7:
From the examples of matches in the task sheet, it seems to me like there should be no expression before the first partial sequence in the search and no expression after the last partial sequence in the search. It is obvious that the search ACTG*GTAC*CA should give a match on ACTGACCGTACTGCA.

My question is: Should it give a match if add any number of character before or after the sequence above? Should GACTGACCGTACTGCAG also give a match?



Thread
# assessment

expression before the first partial sequence in the search and no expression after the last partial sequence in the search. It is obvious that the search ACTG*GTAC*CA should give a match on ACTGACCGTACTGCA.

My question is: Should it give a match if add any number of character before or after the sequence above? Should GACTGACCGTACTGCAG also give a match?

1 reply

James Weeks (tutor) 7 days ago
Yes, it would. The search in level 7 should search the whole sequence for a matching substring

Reason for not using Regex:

After creating the test as shown in the report, I think Level 7 works. I only know about Regex in Week 12 so while Regex is more efficient, I did not implement it for L7.

Self-reflection

1) How do I think I went with this assignment?

I think overall, I completed 90% of the assignment so I think this is good progress.
The other 10% are:

- *Class cohesion could be improved, regarding MetaTag and SequencePair class*
- *Regex could be implemented for L7 for increasing efficiency*
- *Code quality could be improved*

2) What did I find difficult in this assignment?

Without taking Regex into account, I think Level 4 is by far the most difficult level because it requires me to understand offset and Seek(), bytes and Read(). It is difficult in terms of theoretical aspect and coding aspect.
Level 7 is also difficult on the theoretical and coding aspects but I can understand the concepts more easily the offset and bytes for Level 4

3) What would I do differently next time?

I would start sooner on the assignment to improve class cohesion and implementing Regex.

4) Were there any bugs in my assignment, if so what were they?

I think there are no bugs for L4-6 as I tried to test the program frequently on 16S.fasta. However, I could not test L7 directly on the 16S.fasta and within the context of my small 16S_test.fasta, I think there would be bugs for Level 7.

Self-assessment

Code Quality To score points in this section, the student must follow the code quality guidelines as specified in the C# Coding Style Guide on Blackboard	**25/30**
Maintained consistent, clear, and meaningful standard in variable and method naming. No magic numbers.	3/3
Well structured – consistent and appropriate white spacing, line length, indentation, and separation into files within the project (i.e. one class per file)	2/2
Well commented – class header comment at beginning of each class, comment before every method, and in-line comments to explain complex or not easily discernible code. In-line comments are not excessive.	3/4
The DRY principle (Don't repeat yourself) is followed where appropriate	2/3
Methods are single purpose and clear	4/4
Classes are well designed, with high cohesion and low coupling	6/8
Classes are separated into reusable modules where appropriate	2/3
Exceptions are thrown and handled appropriately	3/3

Basic Functionality To score marks in this section, your program must be able to be run from the command line with the appropriate arguments.		**15/15**
Basic Output	The program displays the data from the file	1/1
	The program displays the appropriate line	1/1
	The correct amount of information is displayed, e.g. only the relevant entries	1/1

	The correct level, provided as a command line flag ***-levelN***, is executed	1/1
	The program **does not** store the whole file in memory, instead it accesses the file on disk	5/5
	Total:	**9/9**
Error Handling	A clear error message is displayed when an incorrect number of arguments is provided	2/2
	A clear error message is provided when an incorrect flag is provided (e.g. not -level1, etc.)	2/2
	A clear error message is provided when the input file doesn't exist, or is incorrectly formatted	2/2
	Total:	**6/6**

<u>Part II</u> To score marks in this section, your program must be able to run levels 4-7.		**Marks Available:** **55/55**
Level 4	The program creates a file as specified by the command line arguments	2/2
	The index file contains a list of all the sequence ids with the appropriate byte-offset	5/5
	The searching program makes use of the created index file to execute a number of queries	5/5
	A clear error message is provided when the index file does not exist	1/1
	Clear error messages are provided when the query file cannot be found, or when a bad query is given, like in Level 3	2/2
		15/15

	Total:	
Level 5	The program correctly locates and prints the requested sequence ids	10/10
	A clear error message is provided when the sequence does not exist	5/5
	Total:	**15/15**
Level 6	The program correctly locates and prints the requested sequence ids	15/15
	A clear error message is provided when the keyword does not exist	5/5
	Total:	**20/20**
Level 7 (Optional)	The program correctly decodes the expression given and identifies any matching sequences	5/5
	A clear error message is provided when the sequence does not exist	5/5
	Total:	**10/+10**

Progress Report	Statement of completeness 2.5 Self-reflection 2.5	**5/5**

Self-assessed marks: 105/100

Note: maximum mark in part II is 55, but level 7 has 10 additional marks that can offset lost marks.

<u>Penalties</u>		**Part II Marks Lost:**
If the statement of completeness, self-reflection is incomplete or missing, the student will lose marks		**0/-10**
Statement of Completeness	Missing	0/-7
	Incomplete (up to -6)	0/-6
	Total	**0/-7**
Self-Reflection	Missing	0/-3
	Incomplete	0/-1.5

	Total	**0/-3**

Missing Progress report	**0/-20 from 100**

Self-assessed penalties: 0

Part II final score	**100 (105)/100**