CS2521 — Algorithmic Problem Solving

DNA manipulation

Overview

In this assessment, you will investigate algorithms and data structures related to bioinformatics.

Learning Outcomes

This assessment has the following learning outcomes.

1. Demonstrate an ability to instantiate appropriate algorithms and data structures for a problem.
2. Analyse and understand the effects of different algorithms and data structures on performance.

Overall Contribution

This assessment will count for 25% of the overall mark for the module.

Hand-in

This assessment is due at 23:59:59 on the 30th of April 2021. Handing in up to 24 hours late will attract a 10% penalty, while handing in up to 7 calendar days late will attract a 25% penalty. Both penalties will be deducted as a percentage of the mark obtained. Work handed in more than a week late will be treated as a "no paper".

Submission should take place via myAberdeen. Please upload a single .ZIP file containing your source code. You should include a single report, *as a PDF file* called `repot.pdf` covering tasks 1c, 2a, 2c and 2e. The textual portions of the report should be no longer than 2 pages in length (single column 11pt font and 2cm margins), though you may include any number of graphs you wish on additional pages.

Please adhere to the formatting instructions, as all outputs will be automatically marked. Marks will be deducted if output formats differ, or if submissions are provided in different file formats.

Please use Python 3 for your code, and ensure that you implement all data structures except arrays, sets and dictionaries yourself. You should not need to access any libraries except for `sys, io, math` and `random` (though you may wish to use additional libraries to generate your plots). Please include all files necessary for running your code in your submission. To mark your code I will be running your output file with input as specified below.

Plagiarism

Plagiarism is a serious offence, and will not be tolerated. If you are unsure about whether your work counts as plagiarised, please contact the course coordinator before the submission deadline. For further details, please refer to the Code of Practice on Student Discipline (`https://tinyurl.com/y92xgkq6`).

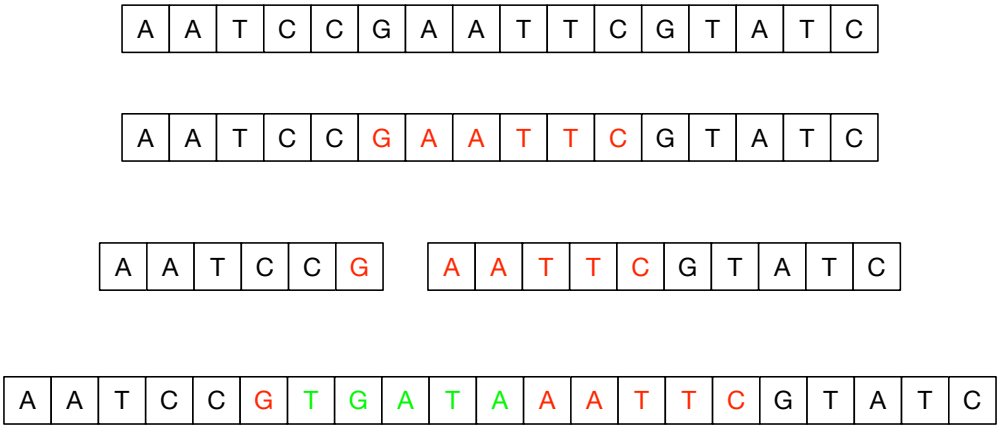

Figure 1: An example of the application of a restriction enzyme GAATTC with index 1, and a spliced in strand TGATA on the original DNA strand AATCCGAATTCGTATC

Assessment Tasks

Task 1 - DNA cleaving

Modern DNA manipulation techniques allow for the insertion of DNA sequences within specific points in a genome. In this task, you will simulate this insertion process.We begin by describing a simple model of DNA.

A strand of DNA is a string from an alphabet of 4 different characters, CTAG. A *restriction enzyme* cuts a strand of DNA at a specific location — the *binding site* — resulting in two separate pieces of DNA. The restriction enzyme specifies the pattern of the binding site, and the point at which the strand then splits. It is then possible to *splice in* a strand of DNA before joining the two pieces. Figure 1 illustrates the process. Here, the original DNA strand was "AATCCGAATTCGTATC". A restriction enzyme "GAATTC" which splits the original strand at index 1 (assuming we count from index 0) will result in two DNA strands, "AATCCG" and "AATTCGTATC". If we splice in the strand "TGATA", the resultant DNA strand will be "AATCCGTGATAAATTCGTATC".

For Tasks 1a and 1b you must adhere to the following input and output format.

Input. Input will be provided via STDIN. It will consist of multiple lines. The first line will consist of two integers M and N, with M denoting the length of the original DNA strand, and N denoting the number of restriction enzymes/splicing operations. The following line will consist of M characters drawn from the alphabet $CTAG$, encoding the original DNA strand. The remaining N lines are of the form $X\ Y\ S_1\ I\ S_2$. Here, X and Y are the lengths of strings S_1 and S_2 respectively, where these are drawn from the alphabet $CTAG$. S_1 is the pattern identifying the binding site, while $I > 0$ is the index where the split will take place. S_2 represents the pattern to be spliced in.

Output. Output is a single line written to STDOUT representing the new DNA strand, terminated by a newline (`\n`).

Example Input.

```
16 1
AATCCGAATTCGTATC
6 5 GAATTC 1 TGATA
```

Example Output.

```
AATCCGTGATAAATTCGTATC
```

If the original strand contains multiple copies of the restriction enzyme, then multiple substitutions will occur. As a simplifying assumption, we will assume these are done sequentially and continue after the inserted string. For example, given the string AAAAAA and the restriction enzyme AAA with index 1, splicing in T, we would get ATATATATAA, and given CAAT with the restriction enzyme A, index 1 and splice of GA, would give CAGAAGAT. Furthermore, if multiple restriction enzymes are provided, you should apply the first enzyme completely before applying the second, third and so on. Once all are applied, there is no need to reapply the first enzyme again.

Task 1a

Using strings, implement the splicing process. Call your file `task1a.py`.

Task 1b

Using a linked list, implement the splicing process. You should provide two separate files for this. The first should perform the search using the naive procedure described in week 2's slides, while the second should perform the search using the hash based method described in week 3's slides. Call your two files `task1bi.py` and `task1bii.py`

Task 1c

Provide plots for runtimes for input DNA strands of length 100000-100000000 (or which take up to 2 minutes to run), with replacement strands of length 10, 100 and 1000 for the string and both linked list based approaches. These plots should be averaged over 20 runs to overcome small timing differences. Make sure that the target strand exists within the input DNA strand at least once. You should provide separate plots for loading the data into the data stucture, the "find" and "insertion" portions of the operation. Attach a short textual discussion describing what the strengths and weaknesses of the different approaches are.

Task 2 - Gene detection

A gene is a strand of DNA that is used — within a cell — to create a protein. Different people share many genes, and one way of detecting a gene is to identify commonalities across DNA strands from multiple people. This task focuses on gene detection, which we think of as the longest contiguous DNA sequence shared across N different DNA strands. Your programs for this task should adhere to the following input and output formats.

Input. Input will be provided on STDIN. The first line of input will contain three integers separated by a space, N,M and L, denoting the number of DNA sequences you will be provided with; the threshold number of strands across which a gene could be detected, and the length of each DNA strand. The remaining N lines of the file will take the form S, a DNA strand. You may assume that $N, M \geq 2$.

Output. As output, write a single string representing the gene to STDOUT, terminated by a newline character (`\n`).

Example. Consider the following input

```
3 2 10
AAAACCCTTG
CCAAAACCTA
ATCCATAACC
```

This requires us to find the longest DNA sequence shared by at least 2 strands, resulting in output

```
AAAACC
```

Task 2a

Assume only 2 DNA strands exist. Provide the pseudo-code for a *recursive* algorithm to find the longest DNA sequence shared by the two strands, and describe its worse case time complexity. Implement it in a file called `task2a.py`

Task 2b

Implement a dynamic programming based solution, assuming only 2 DNA sequences. Call the file `task2b.py`

Task 2c

Provide plots for the runtime for 2 strings, until one of your implementations takes a minute or more to run. Again, your plot should cover the average of multiple runs. Provide a short textual discussion describing what you observe. You may wish to consider issues such as whether the length of matched strands has an effect on runtime.

Task 2d

Extend your dynamic programming approach to handle an arbitrary number of DNA sequences. Call the file implementing this `task2d.py`

Task 2e

Take a look at the paper found here: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=323593`[1], and using the ideas from it, implement a GST based approach to the task. Call the implementation `task2e.py` Provide plots comparing the dynamic programming approach to this approach, and include a short textual description of your observations. Note that this task is potentially *very* challenging.

Marking Scheme

Note that tasks build on each other. You should therefore not attempt later tasks without completing earlier ones; *marks will not be given if earlier tasks are not completed.*

Task	Marking Notes
1a	Completing this task will yield a CGS E3
1b	Completing this task will yield at most a CGS D3
1c	Completing this task will yield at most a CGS C3, based on the depth of observations and quality of analysis.
2a	Completing this task will yield at most a CGS C1
2b	Completing this task wil yield at most a CGS B3
2c	Completing this task will yield at most a CGS B1, based on the depth of observations and quality of analysis.
2d	Completing this task will yield at most a CGS A3
2e	Completing this task will yield at most a CGS A1, based on the depth of observations and quality of analysis.

Changelog

16/04/2021	Cleared up how insertion works
21/01/2021	Initial version

[1] You might have to use the "institutional login" button to get access to the paper from that page.

Acknowledgements

Portions of this assessment were inspired by `http://nifty.stanford.edu/2009/astrachan-dna/`