**504.420 Algorithms for Bioinformatics**

**William McElhenney**

**JHU ID: wmcelhe1**

**Project 3 Analysis**

**Due Date: 12/3/2019**

**Turned In: 12/3/2019**

**Project 3 Analysis**

Analysis of Longest Common Subsequence

What I Learned

When in doubt (and when possible) use outside sources to verify your work. I tried to verify my output by looking for LCS examples on Google. I came across one from Columbia University that provided the wrong answer, which I verified through a source forge LCS calculator (my output matched the calculator).

What I Might Do Differently Next Time

The error check for multiple equals signs is implemented as a *for* loop instead of regular expression as I would like. This is because I could not get a regular expression that would properly detect both equals signs, so next time I would like to spend a little more time to get that properly working.

Design Justification

The project is broken into three files *LCS.py*, *fileIO.py*, and *project3.py*.

*Project3.py* is the driver script of the program. It is in charge of accepting the command-line arguments and coordinating file IO with the LCS solution. It is designed to be as straight forward as possible. It simply takes the input parameters (ensuring they are good) and calls sequence generators before running through all combination of the input sequences (lines 59 – 71).

*LCS.py* is in charge of calculating the length of the Longest Common Subsequence and building the sequence from the matrices generated. *Calc\_lcs()*  (lines 28 – 58) is the implementation of the LCS-Length algorithm on page 394 of our text. I tried to make this as one-to-one as possible with the text, but some changes were made due to differences in start indexing and personal/python quirks. This file also contains the function for building the LCS from the b-matrix that is produced by *calc\_lcs()*, which is called *build\_seq()*. *Build\_seq()* uses the directions encoded in the b-matrix to determine the correct sequence, however it is very different than its book counterpart because I wanted to be able to return the whole string and not just print it out as the function progresses. As such, *build\_seq()* is an iterative solution to building the sequence that by default will run from the bottom-right most cell until it reaches zero either row- or column-wise at which point it will return the sequence for output.

*FileIO.py* contains functions for managing file input and output. It contains one input function, which reads from the command-line parameter specified input file and makes sure it is congruent with what we expect the input data to be. The two output functions are to initialize the output file (*init\_outp()*; lines 98-108) and to append the found and labeled LCS sequence(*write\_outp()*;lines 84 – 89), respectively.

Issues of Efficiency

I do not think there are any issues of efficiency to really be considered here.

Certainly, the storage amount does differ with when chaining, but I don’t think this is really that big of an issue and it is necessary for the proper function of the program.

I think the time complexity should be approximately what is expected from hash tables.