605.620: Algorithms for Bioinformatics

Thomas Muamar

Project 3: Dynamic Lab

Due Date: December 11, 2021

Dated Turned In: December 6, 2021

Justification for data structures/reasons for choices:

For this program I decided to separate the code into two separate classes. One class which is ReadFile basically takes the input file and reads the sequence information by using a java regex pattern. I basically inputted a pattern which would only work for a set of DNA sequences so the letters A, T, G, C were set up to be read from the input file. The reason for this Pattern is because in the input file I didn't want it to read S1 and equal symbol in the text file. Basically, allowed for the text file to be able to keep the labels that identified each sequence. With the pattern regex I used matcher regex in order for it to locate the pattern that was wrote in the code. The items that were read from the input file would then be put into a linked list. The reason I decided to choose a linked list is because the linked list would allow for more space to be allocated. If I were to choose an array I would have to know how much space would need to be allocated. The second method which is the main method in this class would be the main method and this produces the output file if there was an input file name inputted. Goes through a nested for loop to allow the comparison of each sequence with out any repeats or comparing with itself.

The second class contains the main code that would produce the information in the output file which is the longest common Subsequence and the length of this subsequence. In this class there is a 2d array that is used to find the length of the longest common subsequence and then there is a recursive function that will print out the sequence from the 2d array. For the size of the 2D array it contained the length of the string X which is the first sequence and then the length of the String Y which would be the second sequence. Using the 2d array allowed to not have to declare more than one array for the two sequences that we are comparing with each other. The code for the longest common subsequence was basically using the algorithm from the chapter 15.4 reading with some modification in order to make it work for my implementation.

What I might do differently next time:

Something I would like to consider doing differently next time is to have a naïve algorithm to have a comparison side by side to be able to get a better depiction on the run time complexity that is produced by the two algorithms.

What I learned:

One new thing I learned was using the pattern regex in java to be able to read the input file without reading the sequence name. This in combination with matcher allowed to read the whole sequence and ignore the information that was not required for the comparison. Another thing I learned was how to create an LCS from reading in chapter 15 gave me a better understanding for this project and the video that was provided on sequence alignments. Also made me get better understanding on dynamic programming because this was a little bit of a struggle at first.

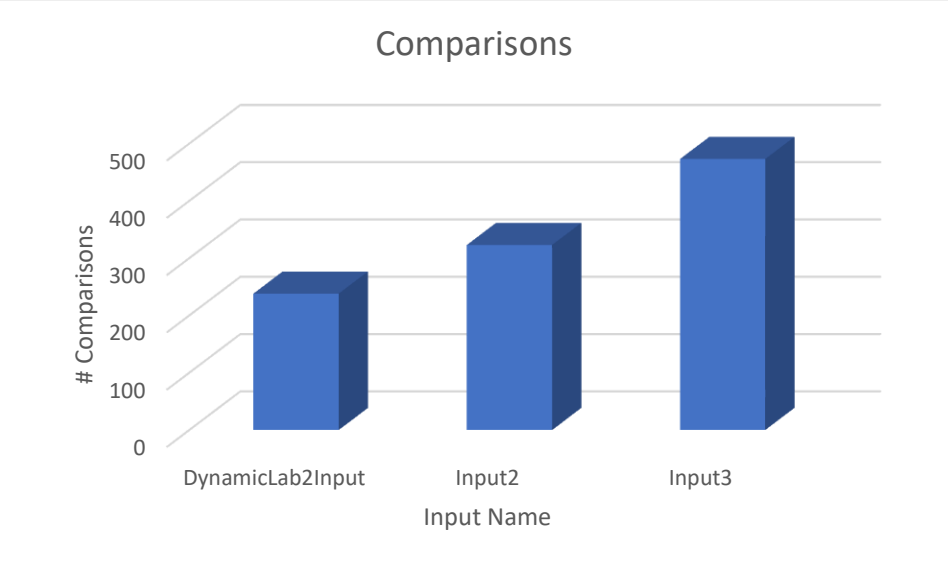
Run time analysis:

In this program for the LCS it contains a double nested for loop in the method for finding the length. Everything from the double nested for loop that is under neath would most likely run-in constant time. So, for this method the first for loop takes into consideration the length of the first sequence while the second for loop ends up taking the length of the second sequence. This would make the asymptotic running time of O(XY). X is the first sequence length and Y is the second sequence length. So, the run time for this program is mainly dependent on the input size that is represented because the number of comparisons ends up increasing as the input size has increased.

Then there is the second method in this that will print out the longest common Subsequence. This method is a recursive function that will increase either I or J by one each time which are the indices. The base case in this recursive function is when either I or j is set equal to 0. So, the complexity for the recursive function would be O(x+y). So, the entire algorithm here for the dynamic program is o(xy) + o(x+y) which is simplified to O(xy) for the total run time.

To confirm that the input size of the sequences was an important depiction for the program I made extra inputs with different sizes. Starting with the initial required input and adding on to the length of each of the sequences for the next inputs.

Name Of Input	#Comparisons
DynamicLab2Input	238
Input2	323
Input3	473



As shown in the graph and table the number of Comparisons seems to increase as the input size has increased. This would mean the running time would increase as the input size increases. The reason for that is because the program would have to make more comparisons to find out the longest common subsequence.

Relevancy to Bioinformatics:

For the longest common subsequence algorithm this would be used in bioinformatics to align sequences. It is important to align sequences because it allows to find similarity which can then help producing this like phylogenetic trees. It is also important in identifying functions of unknown proteins because you can use existing database of proteins to make a comparison with the unknown protein to find similarities between another protein. It can also be applied to DNA sequences and this is important because when

comparing DNA of different organisms you can find identical DNA patterns between these two individual organisms.