605.420: Algorithms for Bioinformatics

J. Beckett

Project 3 Analysis

Project 3 Analysis

Data Structures and Space Complexity

An array list was used to store the sequences as they were read from the input file. An array list was chosen over an array because you will not always know how much space to allocate. A matrix for finding the length of the LCS, on the other hand, was created using a 2D integer array. The size of the array was m+1 x n+1, where m is the size of the first sequence, and n is the size of the second sequence. The matrix was filled with Ints, and ArrayList stores them as integer objects whereas an Array stores it as int, no conversion necessary. Array was the most efficient choice when it came to choosing the structure for the array. To print the longest common subsequence from the matrix, you can create a second matrix of the same size as the first and reference it to determine which cell you should go to next. This gives a total space complexity of $\Theta(mn)$. I opted not use this method to save space since you have all the information you need in the first table. This cuts the space in half, but it does not change the space complexity asymptotically as the space used is still $\Theta(mn)$.

Run-time Analysis

The number of expected comparisons increases the length of the sequences increase. The first method called in the constructor, *createTable(),* fills in the LCS matrix. The method contains a double-nested for loop – the first loop runs the length of the first sequence and the second loop runs the length of the second sequence. Everything underneath the loop runs in $\Theta(1)$, giving this method an asymptotic run-time of $\Theta(mn)$ where m is the length of the first sequence and n is the length of the second. The pseudocode for this method has two loops assigning zeros to the first row and column of the matrix. The default value of an integer array in Java is 0, so the loops were not necessary, though this does not change the asymptotic run-time. The number of comparisons in this method will not change based on the value of the sequences because it visits each cell in the matrix and the matrix is solely dependent on input size. Table 1 attached lists the number of comparisons in *createTable()* and the number of comparisons that actually occurred. They are equal for every value.

The seconded method called in the constructor, *longestCommonSubsequence()*, is a recursive function that calls itself decrementing the indices i and/or j by one each time. Since the method ends when either i or j equals 0, is has a worst-case run-time of $O(m + n)$. Unlike the previous method, this one does not have the same number of comparisons for each value. The run-time stated is for the worst case scenario, so it is expected that the actual number of comparisons is smaller than what is expected. Figure 1 helps illustrate this by comparing the actual number of comparisons to the expected number. There is a linear fit with an R^2 value of 0.9526. As the numbers get larger, the actual number of comparisons will get closer to the expected number.

The combination of these two methods results in a run time of $O(mn)$.

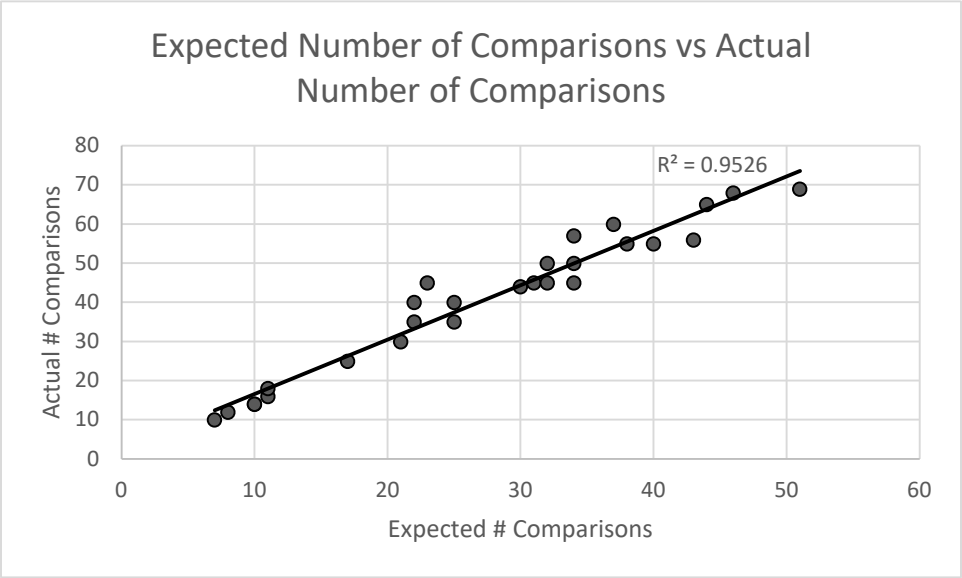

Figure 1: Expected number of comparisons vs the actual number

What I learned

I learned how important algorithms really are in bioinformatics, and how important implementation is. I worked on chapter 32 at the same time as this project, so it was interesting to learn more about string matching and find the LCS at the same time.

What I Would Do Differently/What I Improved Upon

I set up my program so that it will only accepts the values ATCG. It would be very easy to add in "U" if you were matching RNA sequences. If you were matching any string, you could split each row if there is whitespace and only take the last value. This is a way to make sure the string name and equal sign are not added to the final strings that will be used.

Applicability to Bioinformatics.

The longest common subsequence (LCS) algorithm is used in bioinformatics to align sequences. Aligning sequences is used to identify conserved regions of the genome, identify homologous organisms, identify an unknown sequence, and to solve many other problems. Bioinformatics comes with large amounts of data, and saving time and space wherever you can makes a big difference.