Programming Assignment #3

Brendan Foley

26 April 2023

1 Description

In this project, we analyze the performance of finding the longest common substring (LCS) between two strings. To accomplish this, we perform dynamic programming to align the two sequences based on user defined match scores, mismatch penalties, and gap penalties. After aligning, we again perform dynamic programming to find all base matches (diagonal moves in our dynamic programming matrix, starting from the bottom right of our matrix).
Our code makes use of three main functions: get_max, backtrace, and get_LCS. get_max is used to populate the dynamic programming matrix. This method looks at the left, upper-left, and upper cells to see which one will provide the maximum score. backtrace goes backwards through our matrix to find the aligned sequences, adding gaps when we traverse leftward or upward. Finally, get_LCS is used to find the longest common substring by traversing the matrix once again, however, when we traverse on the upper-left diagonal, we record what the character is.

2 Performance

The theoretical complexity of the longest common substring (LCS) algorithm is O(n*m), where n and m are the lengths of the two strings being compared. My algorithm functions in O(n*m) as well. This comes from the creation of the dynamic programming matrix, since we must fill each of the cells in the n-by-m matrix.
The results shown below reflect total number of comparisons of my total program, which includes both backtracing steps. We chose to break up the backtracing into two functions in order to allow for future implementations of alignment when we do not want the LCS. While for this application of my program it results in a slightly longer run time (negligible for the lengths of strings that I have ran), it will provide for selective running of functions down the line.
In our testing, in addition to the required input, I generated strings of lengths up to 900 to compare. I included two edge cases: a single character string, and repeating strings.
Figure 1 shows us a summary of efficiency for varying size of matrices. A noticeable point is the low number of comparisons just above the matrix size of 1250. This is an outlier generated by testing between the edge case of a string of size 1 with a large string. However, the general trend of O(n*m) is appreciable.
Table 1 summarizes the results and statistics associated with the required input provided by the assignment. The results of our extended testing are in the Appendix subsection ”All Comparison Results”, barring the LCS, as some of the strings are too long to reasonably report in this paper.

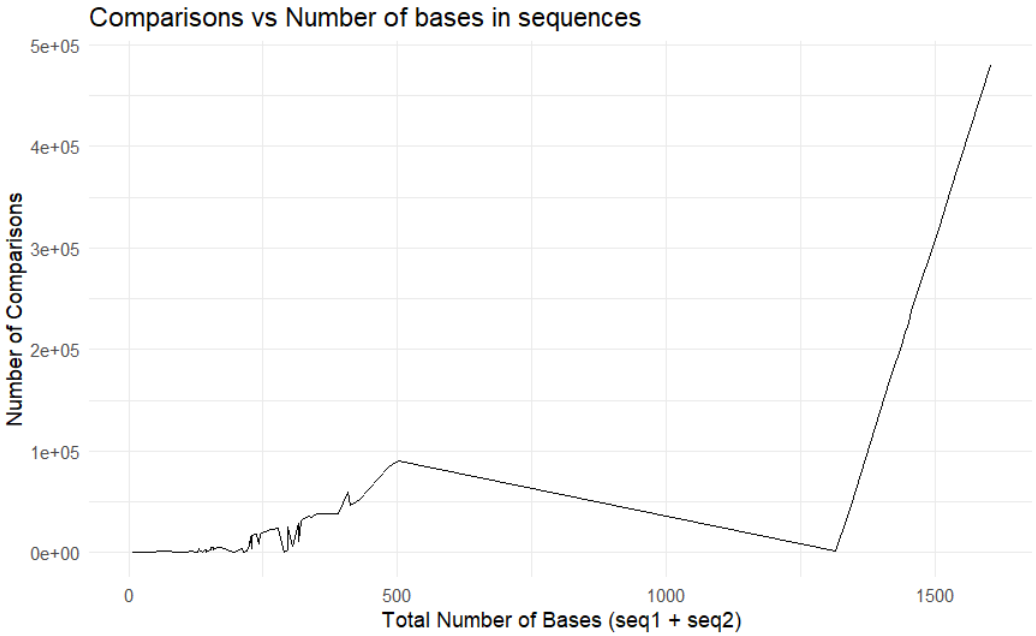

Figure 1: The number of comparisons against total number of characters between both strings.

Number of Comparisons	Number of Bases	Average Seq. Len.	LCS
1113	57	28.5	GCGCGGAAGCCGGCCGAA
1510	69	34.5	AGGCGATGCGCC
1479	68	34.0	GCGGATGTTTT
613	45	22.5	CTGCAAGGA
624	44	22.0	TTGCTTGTA
865	56	28.0	TTGTTAATC

Table 1: The summary of the results from pairwise comparisons of the required input.

3 Analysis

We chose to implement an iterative dynamic programming approach to solve the longest common substring (LCS) problem. To do this, we must use an (n+1)-by-(m+1) matrix, where n and m are lengths of the given strings. As justified previously, we chose to use 2 backtracing steps, one to align the strings followed by one to get the LCS. This allows selective running of methods. We also elected to design a class, Cell, which serves to keep track of the value stored in the dynamic programming matrix, as well as which cell (above, left, or diagonal) the current cell derived its value from. While this may add overhead of storage by implementing a new data structure, it eases the assessment during backtracking, as we retrieve the previous cell location rather than derive it based on penalty/score.

In this project, we learned how to approach the alignment of two sequences through the use of dynamic programming. While we used a local alignment strategy, it would be interesting to next pursue a global alignment strategy, as well as assess how modifying the match, mismatch, and gap scores will effect runtime and alignment, as has been done throughout the literature. Finally, it would also be worthwhile to consider how to tackle the alignment and LCS of n sequences, which will be a common problem in the ”real world”.

4 Application

The application of the longest common substring (LCS) method to bioinformatics is immediately obvious. The bread and butter of a bioinformatician's toolkit is the use of alignment and similarity between several sequences. This has uses in phylogeny, gene/protein functional discovery, and genomic annotation.

5 Improvements

While we did not make any broad improvements over what was tasked in the assignment, we did implement a method for fun that prints the backtracing matrix. This is not reflected in the output, but can be seen and used through an IDE. It was mainly for fun, but was useful in debugging and visualizing the dynamic programming steps.

6 Appendix

6.1 All Comparison Results

Number of Comparisons	Number of Bases	Average Seq. Len.	Comparisons/Base
371	32	16	23.1875
369	32	16	23.0625
388	32	16	24.25
155	23	11.5	13.47826087
156	23	11.5	13.56521739
163	23	11.5	14.17391304
22	17	8.5	2.588235294
23	17	8.5	2.705882353
24	17	8.5	2.823529412
17	8	4	4.25
625	45	22.5	27.77777778
634	45	22.5	28.17777778
620	45	22.5	27.55555556
254	36	18	14.11111111
36	30	15	2.4
624	45	22.5	27.73333333
616	45	22.5	27.37777778
615	45	22.5	27.33333333
246	36	18	13.66666667
35	30	15	2.333333333
1156	58	29	39.86206897
25616	1330	665	38.52030075
25509	1330	665	38.3593985
25640	1330	665	38.55639098
12069	1321	660.5	18.27252082
1724	1315	657.5	2.622053232
46622	1343	671.5	69.42963515
46540	1343	671.5	69.30752048
2067	118	59	35.03389831

Continued on next page

Table 2 – continued from previous page

Number of Comparisons	Number of Bases	Average Seq. Len.	Comparisons/Base
2054	118	59	34.81355932
2077	118	59	35.20338983
890	109	54.5	16.33027523
127	103	51.5	2.466019417
3759	131	65.5	57.38931298
3768	131	65.5	57.52671756
170122	1416	708	240.2853107
2505	141	70.5	35.53191489
2503	141	70.5	35.5035461
2501	141	70.5	35.4751773
1086	132	66	16.45454545
155	126	63	2.46031746
4544	154	77	59.01298701
4538	154	77	58.93506494
207506	1439	719.5	288.4030577
16357	227	113.5	144.1145374
2726	152	76	35.86842105
2713	152	76	35.69736842
2688	152	76	35.36842105
1135	143	71.5	15.87412587
162	137	68.5	2.364963504
4989	165	82.5	60.47272727
5010	165	82.5	60.72727273
226359	1450	725	312.2193103
17746	238	119	149.1260504
21768	261	130.5	166.8045977
2559	144	72	35.54166667
2573	144	72	35.73611111
2570	144	72	35.69444444
1135	135	67.5	16.81481481
162	129	64.5	2.511627907
4679	157	78.5	59.60509554
4662	157	78.5	59.38853503
213780	1442	721	296.5048544
16710	230	115	145.3043478
20547	253	126.5	162.4268775
22320	264	132	169.0909091
2834	158	79	35.87341772
2813	158	79	35.60759494
2836	158	79	35.89873418
1269	149	74.5	17.03355705
181	143	71.5	2.531468531
5173	171	85.5	60.50292398
5171	171	85.5	60.47953216
237869	1456	728	326.7431319
18429	244	122	151.057377
22677	267	133.5	169.8651685

Continued on next page

Table 2 – continued from previous page

Number of Comparisons	Number of Bases	Average Seq. Len.	Comparisons/Base
24481	278	139	176.1223022
23245	270	135	172.1851852
4057	211	105.5	38.4549763
4070	211	105.5	38.57819905
4035	211	105.5	38.2464455
1470	202	101	14.55445545
210	196	98	2.142857143
7561	224	112	67.50892857
7575	224	112	67.63392857
322863	1509	754.5	427.916501
25823	297	148.5	173.8922559
31678	320	160	197.9875
34322	331	165.5	207.3836858
31985	323	161.5	198.0495356
35808	337	168.5	212.5103858
4493	229	114.5	39.24017467
4490	229	114.5	39.2139738
4469	229	114.5	39.03056769
1540	220	110	14
220	214	107	2.056074766
8337	242	121	68.90082645
8355	242	121	69.04958678
353831	1527	763.5	463.4328749
28291	315	157.5	179.6253968
34463	338	169	203.9230769
37686	349	174.5	215.965616
34875	341	170.5	204.5454545
38705	355	177.5	218.056338
59090	408	204	289.6568627
6010	304	152	39.53947368
6022	304	152	39.61842105
6031	304	152	39.67763158
2177	295	147.5	14.75932203
311	289	144.5	2.152249135
11267	317	158.5	71.0851735
11249	317	158.5	70.97160883
479708	1602	801	598.886392
38245	390	195	196.1282051
46540	413	206.5	225.3753027
50452	424	212	237.9811321
47163	416	208	226.7451923
52546	430	215	244.4
83842	483	241.5	347.1718427
90609	501	250.5	361.7125749