Bryan Arnold

CSE 3500

Programming Assignment 2

4/12/17

Settings: The language I used was Java in the Eclipse compiler given by Oracle. My machine is an Asus Laptop with 64-bit operating system (Windows 10), Intel(R) Core™ i7-4710HQ CPU running at 2.50 GHz. My RAM is 16.00 GB and my hard drive has 1.395 TB of memory (with 943 GB remaining available).

LCS

Running Time of LCS: The running time for creating my dynamic programming table was O(n*m). This is because each part of the table, each spot, must be initialized into several lengths of the sharing subsequence. So, each string must be iterated through, each character being evaluated if sharing characters before or above are shared. So, since each string is iterated through once, the run time must go through both lengths n and m, making it O(n*m).

The running time for tracing back through the dynamic programming table and displaying the longest common subsequence is not really needed to look at as it is less than O(m*n) of the prior LCS method. I think that it runs at around O(m +2) or something, but since this method is still part of the same algorithm in the first method, the first method runtime takes priority. This makes the runtime O(n*m) for the algorithm.

Sequences Examples:

This first example is for strings of length 10, with high error counts:

```
First string being compared: AACTCCATG
Second string being compared: TGTCTACCCC
The length of the longest common subsequence of the two strings: 4
The longest common subsequence: CTCC
The time to run LCS on these strings was: 4516 nanoseconds

First string being compared: AACTCCATG
Second string being compared: AATGCCTTCT
The length of the longest common subsequence of the two strings: 6
The longest common subsequence: AATCCT
The time to run LCS on these strings was: 4105 nanoseconds
```

Now, this example is for strings of length 100:

```
First string being compared:
ATCCACACCTTTCCTTTAGTCCTACTGCGAACCTATCTAGGAAACAGCAGACACAGTCTGTATAGACATTGCGTTCA
TAGAACAGCGTT
```

```
Second string being compared:
TTTCCAGGCTCTTCAATAAGGACCTACATCATCCGTACACCTCTCGAGTTTGCACAGTAAATGTCGCACTTTCTTAT
GTTGCGTGCTAATGTTGAAATG
The length of the longest common subsequence of the two strings: 61
The longest common subsequence:
TCCACCTTCTAGCCTACTCGAACCTTCAGGAAAAGCGCACTCTTATGTTGCGTTATGAAAG
The time to run LCS on these strings was: 70203 nanoseconds

First string being compared:
ATCCACACCTTTCCTTTAGTCCTACTGCGAACCTATCTAGGAAACAGCAGACACAGTCTGTATAGACATTGCGTTCA
TAGAACAGCGTT
Second string being compared:
TTCGCACGTACCGTTTTCTTCTAACTCCATACGGAACTCCCTAAGCATTTTTTAGTACAAATGCCAGCTATCCGTCG
TTGCGATCTGAGGGGACTAGTTG
The length of the longest common subsequence of the two strings: 65
The longest common subsequence:
TCCACACCTTTCTTTATCCTACGGAACCTACTAGAAAAGCAGCACGTCGTTGCATTGAGACAGTT
The time to run LCS on these strings was: 68562 nanoseconds
```

As it can be seen, in terms of larger lengths of strings, the runtime will always be longer. This makes sense as the runtime is O(mn), so the lengths being longer will make the runtime longer. In terms of the scaling of my program, it scales well I'd say, as these runs were very few nanoseconds. To see how my scaling went well, here's an example of strings with length of 1000:

```
Second string being compared:
CAGAAGGCCTCCATGGACACTCAACCGTTGAGGAGCTACTGGCCTTAATTCCGCGACTTCTTCTATCGTCACTATGT
TTATACCTATCACACTTGCCAGCCCTCGCTTCAGAGCGCATGGGACAGCATATGACTAATCGGCGGCGGACTCACTC
GGCGCCGCGGCCGCATGGAGACGGCTAAGCCCACCATCTCTTTGTTGTCTGTAGGTCTTAGACTCGAGTCGAACTAC
GAGGTAGTATTTCCAGACAGCAGATTGAACAGATACTATGCGCAGCTGTCAGCACGCATCGTTTAGGGGCGCAGATG
ACAAACTCGGGTGCGGACGGTACACAACGAGTTTCTTGAAAGGAGTATGAGCGGTAGGGAATAGCGGACATTTCCTC
TTACGCTGAATCGAAGCCTCGGTACCTTACGCCTTTGTAACATCAAGATGCTATAGCGCTCTAGGACTACGGTAGCA
GAATCGCACTATCCGTGTCATCATCATTGCATCTCTAGAAAACAGAGGGTCTTCATACACATTCATTACGGACCCCC
TCACAGTGATTGCTCACTATGTCTTCTAAATATAAAGGTCCAGCCAGGAGATACGCAGTAACACAATTATTTAGAAT
AGCTGCCTAGTCCCTTGTCTCATCCCAAAACCGTCTCTGACCCGGGTGTAATGAGGTGGTAGTGAATTACGCCGCGC
CTTGGAGTTTCAAGTTGATGTATTTCGTTTTTGGTATATGGCACCGTTTAAGAGCTGTCAAGGTAAGTCTAGATTAC
CTATTATTGCAAACCGTGCGGTTGGTCTTCAAATGATTCTCACTCATGTCACCCCTAGTCCCCTAAAGCACACGGTG
CCTCTTCAGCCTGTAAACCTATGTGCTCGCCAGCCCTACCATAAATAAAGTGAGACTGGCACCAGACCAGCAACGGA
CATGGAATAAGTTAAGAGTCTCATTACCAAGGTGATGCAGTGAGAGTGGAGGTATGTTTAGCGGATCCATTCGACC
The length of the longest common subsequence of the two strings: 675
The longest common subsequence:
GAACCTCAGGATAACTTAGGGCTCTGCTTAATTCGCGCTTTTCTTCTCACTATGTTAACCTATCACACGCGTCGCTG
ACCAGGGAAGCATATACTAACGCGGGGATCACTCGCCGGCCGCTAGCGCAACATCTCTTTGTTGTCTGTAGGTTAGC
GGTAACACGAGGTAGATCCAGACAGCAGTTGAACAATCAGCGCAGTGTGACGCATGTTTCGAGTGACAAGGGGCAGA
AAGAGTCTTGAGAGATGAGCGTAGGGAATAGCGGAATTTCTCGCTGCGAAGCTGACCTTGCCTTGTACATCAAATGT
ATAGCCCTAACTACAGCGTCGATGTGACTCATTTCTTAGACGATCCACACATCATTCACTCAAGTGATCTCATATTT
TCTAATTAAAGTCCCCAGAGACCATACACATTTAGATCTGCCATCCTTCTCACCCCCGCTCGCGTAAGAGGAGTGAA
TTAGCGGCCTTGAGTTTCGTTGTGTTCGTTGAAGCAGTTAAGACTGGTAAGTAATTCCATTTTCAAGGCGGTGTAAA
TGATCACATGTCACAGTAAAGCCACGTGCCTCTTCGCCTGTAACCAGTGCCCCAGCACCAAAATAAAGTGAGACGGC
CCAGACCAGCAACACATGGAAGTTGTCTCATACAGAGCAGTGGTGAGAGTAGGGTACCC
The time to run LCS on these strings was: 6855307 nanoseconds

First string being compared:
GGAAATTCCTCGAAGGATAATCTTAGGGCTTCCTGACTTTGAAATTCGCGGCGTTTATCTGTCCCTTACGACCTCAT
```

```
GTCTCAACCCGGTATCTACACGCTGTACGGCGTGATACCGAGGGTAATCTCGAAAACAACTTATCACCTAAACTGTC
CGGGAATGGATTAACCACTCAAGCCGGACCGCCTCAGCGCCAACAGTTCTCCTTCTGTATGGGGGTGACTGTTAGTG
TTCAGCGGAATAAGCACGATGTGGCTAGATCCCAGTACCATAATGTCACGTGTCGACCACAATCCAGCGTTCAGATA
GTGAACCGGTGTGCATGCCTATTCGATGTGTGACCAGAGGGGCAGAAAGAAGGTCTTGTACGAGCAATGAGGCCTGC
TTAAGATCGCGCCAATCACGACGGGTAATATTCTCGCTGCCGTATTAAGACTGACCATTTTGCCGTTGTACATAGAC
AAACTATAGTATTAGTCCCTAACTAACAGCTGTCAGATGTGACTCATTTCGGGTTAGATGCGTATGCCCCACGACAT
CGCAGTTCACATCAAAGAGGTGATCTCATATTTTGCGTAATCTTAACAGTCTCTCCAGTAGGACCATACTGGACATT
TAGATCTGGCGGCAATGGCCTTCAAATACACAGCCGCCGCTCGCAGTCACAGAGGAAGCATGAATTAGACGGCTCAT
ATAGAGTTTGCCGTCCTCCTGGGTGTTACGTATGACAGCAGCTATAATGAACCTGGCTAAAGTAATTTCCATGTTCT
CAAGGAGCGGTAGTAAATCGTAAGTCGACAGTCGGTACGGACAGTAACAGCCTACGTCATGCCTCTTCTGGCGACCT
GGTTTATTTTACCGAGTGGCCCACTCAGAGCAAACCCAAGAACGCTGCAAAAGTTGAGGACGGGGGCGCCCAGACAC
CAGAGCAACCCACAGTGGACAGCGACTTGTCATGCGACTAACAGCACGACAAGTGGTGCAAAAGAGTAGGGTACCC
Second string being compared:
CCATCTGCCTACGACTTCAGTGGTTAATCCCTAGTTTCGTTCGGACATGTATCCGAAGCCGGCCCCGGTCCTCCGGT
CATGCAGATCTCCGATAGCGATTACCCGCTAGGCACACGGGCGCACCATGAGGATGCGGAACCACGCTATGTTTCGG
TTCCGGATTACATTGGCGCACACCGGACGGAGGACGCAGCTCGGCGAACCAGGCCTGTTTGTTATAAAGTATTTCTA
CAGTCATTCGAGAGTGATCGCAGAACATCAAATGAGACTCGGCGCATGGCATTCACCTGAATCTATTGTGATAACGG
ATAAGCGGGACTAGTGTGTGGTCGCGTTGCAAGGACTTTAACATCGAAGAAGGTGCGCGTCACTAAGCGAGAAAGTT
CCAAAGAAACGCCACTCTATAGGCGCTTACTATACCCCGCTCACCACAGAAAAGTTCGTTAATTCAGGTAAACTGGG
TAAATTAGTCATCTGCCCCGCTCAGCATGTGCTCCCACGATATACAATGTCCGTTTGAAACTGACTAGGGCTCCATT
ACGTTCAAATACGATTTCGCTGATCCAAGTGGACACATAAAGACGGACAAGGTCTCTGAGCGCAGGTCTTCGACTTT
TGTCGTATTACATCTTTCATTCCGGTGCAGGGTCGGGCAACCCGGCCAGTCCCGGATGAATAACGGCCTATACTTAG
ATTCTCTGGTTAGTCCCTGCGGAAGGTTGTGAAGTATTGAAAGGTCCACGCTTAATGTCCAGCAGCCCGCTGAGGGT
TATATTATTCGGATTTGGTAATCCGCGCGCCATGGTCTTCACTACATCCATAACCCGGGCGCGTATTCAAACCCAAC
ATTCGATCTCGCGCTCCCGACACTTGTTAGACCCGTTCGTTTGCGCCCGATGAAGTGCCGAATAAATGATAGATTAT
CATGCAGCGACACCAGCCATACCATAGGGACAGCTTAGAGTCGCATCGGGCTTCTCAGGGTGATGTTCAAGGGTCT
The length of the longest common subsequence of the two strings: 659
The longest common subsequence:
ATTCCTCGAAGGTAATCTAGGTTCGACTTTGAAGCGGCCGTCCTCGTCATGCAACCCGTACTACCGCTGACGGCGAC
CGAGGATCGAACCACCTATGTCGGTGGATTAACCACACCGGACGAGCGCAGCTCCGAGGCTGTTTGTTATAAAGATT
CTACAGTCATTGAGTGTCGCAAACATCAAATGAACCGGGTGCATCCTATTATTGTGAAAGGAAAGAAGGTTGTCGGC
AAGACTTTAAATCGGAATCCGACGGGAAATTCCGCGCCATTAAGCGCTTTTCCGTACAAGAAAATTGTATTAGTAAA
CTGTAATTGTCATTCGTAGATGTGCCCCACGAATCAGTCCTAAAGAGGGCTCATATTCAATCTTCGCTTCCAGTGGA
CCATACGGACAAGTCTGGCGCAGGCCTTCATACACACCGGTGCAGTCCAAGGAGCATGAATAACGGCTATATAGATT
TCGTCCCTGGGTGTAGTATGAAGCAGCTTAATGAACCGCTAGTAATTTCATGTTCCGGGCGGTTTCTAATCCATCGG
CGGACAAACCCACTCATGCCTCCCGACCTGTTAACCGGTGGCCCATGAGCAAAAAGAAGATTAAGCGCGACACCAGC
AACCAAGGGACAGCTTGTCGCATCGCCCAGGGTGAGTAGGGTC
The time to run LCS on these strings was: 9027920 nanoseconds
```

Looking at the runtimes of these, they are a lot larger in terms of nanoseconds. But when converting them to seconds, neither of them even reach 0.01 seconds. So, the scaling of input size makes the runtime much larger, but it doesn't blow the runtime through the roof, showing good algorithm scaling relative to string length sizes.

Some observations I saw while running the LCS on these strings, was the runtime of solutions of the same length had different runtimes. The difference was very minimal, so it is probably some random source of error in the processor. Another observation is that the runtime got larger the larger the common subsequence was. This is probably due to the fact more operations are required to happen when there are more matching parts of the sequence, so this also explains why comparing a string to itself results in the longest runtime, which can be seen here for strings of length 1000:

```
First string being compared:
GGAAATTCCTCGAAGGATAATCTTAGGGCTTCCTGACTTTGAAATTCGCGGCGTTTATCTGTCCCTTACGACCTCAT
```

```
GTCTCAACCCGGTATCTACACGCTGTACGGCGTGATACCGAGGGTAATCTCGAAAACAACTTATCACCTAAACTGTC
CGGGAATGGATTAACCACTCAAGCCGGACCGCCTCAGCGCCAACAGTTCTCCTTCTGTATGGGGGTGACTGTTAGTG
TTCAGCGGAATAAGCACGATGTGGCTAGATCCCAGTACCATAATGTCACGTGTCGACCACAATCCAGCGTTCAGATA
GTGAACCGGTGTGCATGCCTATTCGATGTGTGACCAGAGGGGCAGAAAGAAGGTCTTGTACGAGCAATGAGGCCTGC
TTAAGATCGCGCCAATCACGACGGGTAATATTCTCGCTGCCGTATTAAGACTGACCATTTTGCCGTTGTACATAGAC
AAACTATAGTATTAGTCCCTAACTAACAGCTGTCAGATGTGACTCATTTCGGGTTAGATGCGTATGCCCCACGACAT
CGCAGTTCACATCAAAGAGGTGATCTCATATTTTGCGTAATCTTAACAGTCTCTCCAGTAGGACCATACTGGACATT
TAGATCTGGCGGCAATGGCCTTCAAATACACAGCCGCCGCTCGCAGTCACAGAGGAAGCATGAATTAGACGGCTCAT
ATAGAGTTTGCCGTCCTCCTGGGTGTTACGTATGACAGCAGCTATAATGAACCTGGCTAAAGTAATTTCCATGTTCT
CAAGGAGCGGTAGTAAATCGTAAGTCGACAGTCGGTACGGACAGTAACAGCCTACGTCATGCCTCTTCTGGCGACCT
GGTTTATTTTACCGAGTGGCCCACTCAGAGCAAACCCAAGAACGCTGCAAAAGTTGAGGACGGGGGCGCCCAGACAC
CAGAGCAACCCACAGTGGACAGCGACTTGTCATGCGACTAACAGCACGACAAGTGGTGCAAAAGAGTAGGGTACCC
Second string being compared:
GGAAATTCCTCGAAGGATAATCTTAGGGCTTCCTGACTTTGAAATTCGCGGCGTTTATCTGTCCCTTACGACCTCAT
GTCTCAACCCGGTATCTACACGCTGTACGGCGTGATACCGAGGGTAATCTCGAAAACAACTTATCACCTAAACTGTC
CGGGAATGGATTAACCACTCAAGCCGGACCGCCTCAGCGCCAACAGTTCTCCTTCTGTATGGGGGTGACTGTTAGTG
TTCAGCGGAATAAGCACGATGTGGCTAGATCCCAGTACCATAATGTCACGTGTCGACCACAATCCAGCGTTCAGATA
GTGAACCGGTGTGCATGCCTATTCGATGTGTGACCAGAGGGGCAGAAAGAAGGTCTTGTACGAGCAATGAGGCCTGC
TTAAGATCGCGCCAATCACGACGGGTAATATTCTCGCTGCCGTATTAAGACTGACCATTTTGCCGTTGTACATAGAC
AAACTATAGTATTAGTCCCTAACTAACAGCTGTCAGATGTGACTCATTTCGGGTTAGATGCGTATGCCCCACGACAT
CGCAGTTCACATCAAAGAGGTGATCTCATATTTTGCGTAATCTTAACAGTCTCTCCAGTAGGACCATACTGGACATT
TAGATCTGGCGGCAATGGCCTTCAAATACACAGCCGCCGCTCGCAGTCACAGAGGAAGCATGAATTAGACGGCTCAT
ATAGAGTTTGCCGTCCTCCTGGGTGTTACGTATGACAGCAGCTATAATGAACCTGGCTAAAGTAATTTCCATGTTCT
CAAGGAGCGGTAGTAAATCGTAAGTCGACAGTCGGTACGGACAGTAACAGCCTACGTCATGCCTCTTCTGGCGACCT
GGTTTATTTTACCGAGTGGCCCACTCAGAGCAAACCCAAGAACGCTGCAAAAGTTGAGGACGGGGGCGCCCAGACAC
CAGAGCAACCCACAGTGGACAGCGACTTGTCATGCGACTAACAGCACGACAAGTGGTGCAAAAGAGTAGGGTACCC
The length of the longest common subsequence of the two strings: 1000
The longest common subsequence:
GGAAATTCCTCGAAGGATAATCTTAGGGCTTCCTGACTTTGAAATTCGCGGCGTTTATCTGTCCCTTACGACCTCAT
GTCTCAACCCGGTATCTACACGCTGTACGGCGTGATACCGAGGGTAATCTCGAAAACAACTTATCACCTAAACTGTC
CGGGAATGGATTAACCACTCAAGCCGGACCGCCTCAGCGCCAACAGTTCTCCTTCTGTATGGGGGTGACTGTTAGTG
TTCAGCGGAATAAGCACGATGTGGCTAGATCCCAGTACCATAATGTCACGTGTCGACCACAATCCAGCGTTCAGATA
GTGAACCGGTGTGCATGCCTATTCGATGTGTGACCAGAGGGGCAGAAAGAAGGTCTTGTACGAGCAATGAGGCCTGC
TTAAGATCGCGCCAATCACGACGGGTAATATTCTCGCTGCCGTATTAAGACTGACCATTTTGCCGTTGTACATAGAC
AAACTATAGTATTAGTCCCTAACTAACAGCTGTCAGATGTGACTCATTTCGGGTTAGATGCGTATGCCCCACGACAT
CGCAGTTCACATCAAAGAGGTGATCTCATATTTTGCGTAATCTTAACAGTCTCTCCAGTAGGACCATACTGGACATT
TAGATCTGGCGGCAATGGCCTTCAAATACACAGCCGCCGCTCGCAGTCACAGAGGAAGCATGAATTAGACGGCTCAT
ATAGAGTTTGCCGTCCTCCTGGGTGTTACGTATGACAGCAGCTATAATGAACCTGGCTAAAGTAATTTCCATGTTCT
CAAGGAGCGGTAGTAAATCGTAAGTCGACAGTCGGTACGGACAGTAACAGCCTACGTCATGCCTCTTCTGGCGACCT
GGTTTATTTTACCGAGTGGCCCACTCAGAGCAAACCCAAGAACGCTGCAAAAGTTGAGGACGGGGGCGCCCAGACAC
CAGAGCAACCCACAGTGGACAGCGACTTGTCATGCGACTAACAGCACGACAAGTGGTGCAAAAGAGTAGGGTACCC
The time to run LCS on these strings was: 6012045 nanoseconds
```

Overall, these were the only observations that really stuck out to me while looking at the results and making the program.

Edit Distance

Running time of Edit Distance: My algorithm for edit distance runs at O(m*n) time as well. To construct the dynamic programming table, the same process, almost, is done as LCS. It goes spot to spot, evaluating what should be put in place in accordance to what is above, to the left, or below the current spot. After each spot is put into place, the last else statement will make sure that the bottom right will be the minimum, this is a fact since the if statements assure this. Once

the solution is found at the end of the table, the spot value is displayed. This is m*n, the lengths of the two strings, and since both strings are iterated through once, the runtime comes out to O(m*n). The displayed integer is the number of errors/optimal solutions for changing the first string into the second string.

Sequence Examples

First, let us look at strings with length 10, with low error count:

```
First string to be converted: TACCTC
Second string for first string to be converted to: TTCTCTCT
The number of operations to convert string 1 to string 2 is: 3
The runtime to convert string 1 into string 2 is: 821 nanoseconds.

First string to be converted: TACCTC
Second string for first string to be converted to: TACTCTCT
The number of operations to convert string 1 to string 2 is: 2
The runtime to convert string 1 into string 2 is: 821 nanoseconds.
```

Now, this example is for strings with length 100, with low error count:

```
First string to be converted:
AATACCAAGACCTTCTCCTAAGGCATACGCGAACCACATTGCTTAAGTAAGGCGCACTTTCCGTATGAAGTTTGTCC
GTTCATTGGTGAAAGGATG
Second string for first string to be converted to:
TTCCAACGCCTTTCTCCCTATAGTCCTGCCGCACCACCCTTAGCTTATAAGCATTTCCGCTCTCTCCGTGTATGGAT
CTTGTCCGTCTATTATGTAAATG
The number of operations to convert string 1 to string 2 is: 37
The runtime to convert string 1 into string 2 is: 86215 nanoseconds.

First string to be converted:
AATACCAAGACCTTCTCCTAAGGCATACGCGAACCACATTGCTTAAGTAAGGCGCACTTTCCGTATGAAGTTTGTCC
GTTCATTGGTGAAAGGATG
Second string for first string to be converted to:
AATTGCAACCTTGTCTCTGACGTCCTATCGCAACCACGCTTAGCCTAAGTAATCCGCACTACTCCGTGTAGAACTTG
TCGTACTATTGTTGTAAAGATG
The number of operations to convert string 1 to string 2 is: 30
The runtime to convert string 1 into string 2 is: 84162 nanoseconds.
```

Looking at the runtimes of both the 10 and 100 length strings, the runtimes for the 100 length strings are longer. This makes sense since the runtime is O(m*n), so the longer the string lengths the longer the runtime will be. In terms of this program, the scaling is also good. Here is another example for strings that are length of 1000 for low error rate:

```
First string to be converted:
CATATTCCAGGTACCTCTTGAGGGTGCTCTTACTCTGGACCTGACATTGCGCGTCGTTTACCTGCCTCTGTGCATGA
CGTTCTAGCCGATTACACGCTTGGTGCGTTTGCGCCGTACTGAGGATGTGGACGCACTTATACTACGTCCGGGAGTA
CTTATAAACTCTAACGGGGGACGCCCCGGGGTAAACCAGGCCCTCCGTTCTTATGGTTACTGATTCTGGTCTCAGTC
TTCATATCGGAGGTGTGTATCCAGAACCAAAATGTCATGATGACCATCGCGTACAGCTTAGTTGGTTCTTAAGGTGC
ATGGCGAACTATCGTGTGTGCGGCGCAGAAAGGAAGATCTTCGATAACATGATAGCCTGCCGCAAACTTAGCGGGAA
TTCCGCGTCAGCGCGCTATTAATTTAGCGCTGCAACTATTGACCTTCGGAGATCTCGATAAACATGTATAGCCCCTG
ACTCGTCAGCTGTGACCCGCTCGGTTACATCGGATCTTAGAGAGATGGAGCCCCATCGCATTTCCGTCCTAAGATGG
CTCATCCGCTGACCATCAAAGTTTCTCAAGGCCCACGGATTCAACGGCAATTTGAGACTGTCTGCCCTAACCTAGTC
ATCATCGCCACCAAGTTCCGACCCGGGGCGGCCGGAGATGAATTAAACGGCCATTGCAGGATTCATGATGTCTCTGC
GATCGTTTGGTAGTAATGACGCGAGGCTTATGCATAGCCCCTGCTAGTAGCATCATACGAGTGTCATCGACACCGCG
```

```
AGTGCGCCTCCACTACCATCGGTAGGACATTACATACACCTATTCACGCTCCCCTTCGCGAGCCTTGCTTAAACACC
GGTTGGAGCATAGAAACGCTTTCAACACCAATGGAATAAGCGCATAGAGTGACACCAAGCATGCAACCATGTGGGAG
AAGCGTTACGAGTACAGATGAACGTACAAGCGGCGGAGGTTGTCCAGGGATATCGATCCCCCATTAT
Second string for first string to be converted to:
CAATCCCGTACCTAATCTTGAGGTGTGCTTTTCTCGGACCCTACGAGCTTCGCGCGTTCTTACCTGGTCCTCCCGTA
CCTGTCAATTTCATAGCGATTTACACGCTGTTGCGTCCGGATACTGAGGTATTGTGAACCACACTTTCACTTCTCGG
AGGAGTACGTAGTAACTCGACGGGGGCGGCTACCGCGAACAGCAGGGCGCCGCGTTCTATTATGTGGCAAGTCCTGC
ATACATCGAATCGCAGGATGTGTATCACAGTAGCAAATGTCAATTGCACACAGCCTGCAGCTTCAGCTGAATCATTG
CTAACGGGTGCAGTGGGAACCTATCGTTGTCGACGCGGCTATAGAGATCGTTGCATAACATAGAATCGAGCGTGAAT
GCAATTGCGTGAATTCACGCGCTACAATTTCGCGCTTACCATGGACCTTTGGGACGACAACGATAACTTATATAGAT
CCCTTGCACTCGCCAGGCAGCTTGACTCAGTCGGTTACATTCGCCTTGAGTAGTGCAGCCCCTCACGTATTCCGTCT
CGAAGATGATTGCTCTATCCGGCTGATGTCTAAAGCTTTTAAGTTCTCACGCTAGATCTCTAGACATGGGACCTTTT
AGTTATTGTCTGCAATAATTAGTCTCCAGCCCCGAAGTCCGACCGGCCGTCCGGAATGAATTAACGGCCATTGTAAT
CATGAGTTGTGCTCTGGTTCTGGTAGATGGCACCGAGCTTATGCAGCCCGCCAGGAGTCACACTATACGAGTGTATC
CGCCCGCTTGTTGGGTACTCCACGTACATCTACCACGGCACGGCATTTACACTCACATCGTCAACAACGAATCCCCT
TCGCGCACCTATGTTTAAACACTCGTTTGGGCATGACGACACTACCAACACAAGGTAAATTAAGTCGGAGGCGAGCG
AAAAGCGCAACCCCTTGAGAGCGTTGTGACAGCTATAATTCTTTCAGCGAGCGGAGGTTGGTCAAAGGATCATTCG
The number of operations to convert string 1 to string 2 is: 356
The runtime to convert string 1 into string 2 is: 8550865 nanoseconds.

First string to be converted:
CATATTCCAGGTACCTCTTGAGGGTGCTCTTACTCTGGACCTGACATTGCGCGTCGTTTACCTGCCTCTGTGCATGA
CGTTCTAGCCGATTACACGCTTGGTGCGTTTGCGCCGTACTGAGGATGTGGACGCACTTATACTACGTCCGGGAGTA
CTTATAAACTCTAACGGGGGACGCCCCGGGGTAAACCAGGCCCTCCGTTCTTATGGTTACTGATTCTGGTCTCAGTC
TTCATATCGGAGGTGTGTATCCAGAACCAAAATGTCATGATGACCATCGCGTACAGCTTAGTTGGTTCTTAAGGTGC
ATGGCGAACTATCGTGTGTGCGGCGCAGAAAGGAAGATCTTCGATAACATGATAGCCTGCCGCAAACTTAGCGGGAA
TTCCGCGTCAGCGCGCTATTAATTTAGCGCTGCAACTATTGACCTTCGGAGATCTCGATAAACATGTATAGCCCCTG
ACTCGTCAGCTGTGACCCGCTCGGTTACATCGGATCTTAGAGAGATGGAGCCCCATCGCATTTCCGTCCTAAGATGG
CTCATCCGCTGACCATCAAAGTTTCTCAAGGCCCACGGATTCAACGGCAATTTGAGACTGTCTGCCCTAACCTAGTC
ATCATCGCCACCAAGTTCCGACCCGGGGCGGCCGGAGATGAATTAAACGGCCATTGCAGGATTCATGATGTCTCTGC
GATCGTTTGGTAGTAATGACGCGAGGCTTATGCATAGCCCCTGCTAGTAGCATCATACGAGTGTCATCGACACCGCG
AGTGCGCCTCCACTACCATCGGTAGGACATTACATACACCTATTCACGCTCCCCTTCGCGAGCCTTGCTTAAACACC
GGTTGGAGCATAGAAACGCTTTCAACACCAATGGAATAAGCGCATAGAGTGACACCAAGCATGCAACCATGTGGGAG
AAGCGTTACGAGTACAGATGAACGTACAAGCGGCGGAGGTTGTCCAGGGATATCGATCCCCCATTAT
Second string for first string to be converted to:
CATATTTCCGTACCCTAATATTAGGGTGCTTTCTCGGACCGTAATTGGCCGGCCGTTTAGCCTGTCTCCGTACCTTG
ACTGTACTAGCCGATTCCACGGGTTCTCCCGCGTACTGATGATAAAATGTGGAACACACCTTATGCTATCGTCCGAG
GATTATCTTATTAAACAGACGGAACGCATACCGGCTAACCAGGCCTCCGTTCTTATTGTCTCTTTGGTCCTACGTCT
AATCGCAGCATTTGAATCCAGAAGCAAAGTCAATAGTTGACCTCGAGTTCTGCTGAATCTTGTCAACGGGTGCATGG
GAACACCGTGGTGTCGCAGCGGCACAAAGCGATCATCGAGGCACCTATGCAGCGAGTTTGTGCTGCAACTTTAGTCG
AAACATATGCCAGCGAGCGGCTAATATTCGCGCTGCGTAGCTGTTGACCCTTCGCGGGTCGATTCGGGAAAATAGAA
GTCTCTGACTCCAGGGGCGCTGTTGATCGGCTACATACGTACCTTGGACTGAGACCCACACATTTCCGTCCAAAAGA
TGATGCTTATCGCACTGACACTTAAGTTCGCTCAAGCCCAAAGATTTCTAGAACTTGTGCCAAGTTAGCTATGTATT
GCTCGTAACGAGTGCTTTACATCCCACGAAAGTCGACCGTGCATTTTCCGGAGAGTTGAGTATAACGGCGCATTGTA
GATTCATGATGTGGTCTCGGTGTTGGTTACATGGCAGCCGACTGCTAATGCACCCTAGCTAACAATACCACACGCGT
GTCTAGCCCGCGCGGATGGTCTCACTACACTACTCGGTACCATGCACTACAGTTCAAGCTTCACACGCCCCGGCCTC
CAGCGACCTACGTTATTAACACCGTTTTGGGCATCGAAAGCTACTAACAAATGGAAACCAGTCGGACGAGCGGAGCG
ACACCAACAGGCAACCATAGTGGAGAGATGTAGACAGATCTGAACTTCGCAAGGGTTAGGTGGTCAAGGGCTCATG
The number of operations to convert string 1 to string 2 is: 366
The runtime to convert string 1 into string 2 is: 8708105 nanoseconds.
```

It’s expected that the runtimes will be larger, which is the case here, but even though the nanoseconds are much larger than strings of length 100, if the nanoseconds are converted into seconds it is still very fast. Neither of these runs breaks 0.013 seconds. So, once again, the

runtimes get much larger the bigger the string lengths, but the runtimes are by no means too big to still be very efficient.

Some observations I saw while running the edit distance on these strings, was the runtime of solutions of the same length had different runtimes. The difference was very minimal, so it is probably some random source of error in the processor. Another observation is that the runtime got larger the larger the more errors/optimal operations were needed to convert one string to the next. This is probably due to the fact more operations are required to happen when there are more optimal errors to change, so this also explains why comparing a string to itself results in the lowest runtime og a given length, which can be seen here for strings of length 1000 with high error:

```
First string to be converted:
CATTCTCCTCTATATTGATAGGTGGCTTTCGTGTGCCAGATTTGATTCAACTCACCCGTCTTACTGGTCCGCGCCGT
ACCCCTAAACTGGTAAGGATACTCACTCCGGATGGGACCGTTTAAATGAGTGTACCTGGACACCCCTATGAACCGAC
TCGGGATGACCTATTAGATCACGTCGCTGGGGACGAAAGCTATAACGGTAACAGGACGAGTTACGTGTAGTGTCATC
TCAATCTAACGTGCAGCGATGTGTGAACGAATTGTATGTCATTAGACCCTGGAGTCAGCGCATCTGTCGGCCGGCAA
TGTCTCCTATTGTGTCCCCGCCGCGGCCCAAGAATTGGATGACAAGGACACCGTGTAGCTCTTATTGCTGAGACAAA
TCACACTAGCCGCTCCTCGAGGCTACGTCTACCAGGCCACTCAGCGGAGTCTGTTGAGAACCGAAGTTCACCCTTGA
CCTTTCGGGGAAGCATGGAACCCGGTTCTCGATTGGAGAATAAGCCGCTCACGCATTCCGTTCAAATGATAGCGCTT
ATCCGCTGGACTATCCTAAAGGTTCTCGAAGTCAAATTGTTACAACTACCATATCTGGTAGCATCTTATTGCTCCCT
AGCTTACTTCATCCCATCCAAAGTGCGCGCCGCGGCAATTCCGGAGAGATATGAACGGTCAGTTTAGCATTGCTAGA
TGTTGTCGGTGTGGGTAGATGGCCAGCTAACGCAATATGCCTCGGCCGGAGTACACTATTAGCAGTCTCAATTACGC
CCTGCTTCGTTGGTCTCAGATCTATATCGTACTTCGCATACACAACACGAGGCTGCAGCTGCTCCGATAGAACCCTG
CTGTGTTTTAACACCCCGTCTGAGTACATTGAACTATAGCTCAGCTTGACAAATAGGAGGATAGTCGGCACACCGCG
AGACCAGATGAATGTCAACCATCGTGAGAGGTCTTTGGTAAGAAACAATAGATCGTCTGGGCTGGTTCTTTTCGGT
Second string for first string to be converted to:
CATTCTCCTCTATATTGATAGGTGGCTTTCGTGTGCCAGATTTGATTCAACTCACCCGTCTTACTGGTCCGCGCCGT
ACCCCTAAACTGGTAAGGATACTCACTCCGGATGGGACCGTTTAAATGAGTGTACCTGGACACCCCTATGAACCGAC
TCGGGATGACCTATTAGATCACGTCGCTGGGGACGAAAGCTATAACGGTAACAGGACGAGTTACGTGTAGTGTCATC
TCAATCTAACGTGCAGCGATGTGTGAACGAATTGTATGTCATTAGACCCTGGAGTCAGCGCATCTGTCGGCCGGCAA
TGTCTCCTATTGTGTCCCCGCCGCGGCCCAAGAATTGGATGACAAGGACACCGTGTAGCTCTTATTGCTGAGACAAA
TCACACTAGCCGCTCCTCGAGGCTACGTCTACCAGGCCACTCAGCGGAGTCTGTTGAGAACCGAAGTTCACCCTTGA
CCTTTCGGGGAAGCATGGAACCCGGTTCTCGATTGGAGAATAAGCCGCTCACGCATTCCGTTCAAATGATAGCGCTT
ATCCGCTGGACTATCCTAAAGGTTCTCGAAGTCAAATTGTTACAACTACCATATCTGGTAGCATCTTATTGCTCCCT
AGCTTACTTCATCCCATCCAAAGTGCGCGCCGCGGCAATTCCGGAGAGATATGAACGGTCAGTTTAGCATTGCTAGA
TGTTGTCGGTGTGGGTAGATGGCCAGCTAACGCAATATGCCTCGGCCGGAGTACACTATTAGCAGTCTCAATTACGC
CCTGCTTCGTTGGTCTCAGATCTATATCGTACTTCGCATACACAACACGAGGCTGCAGCTGCTCCGATAGAACCCTG
CTGTGTTTTAACACCCCGTCTGAGTACATTGAACTATAGCTCAGCTTGACAAATAGGAGGATAGTCGGCACACCGCG
AGACCAGATGAATGTCAACCATCGTGAGAGGTCTTTGGTAAGAAACAATAGATCGTCTGGGCTGGTTCTTTTCGGT
The number of operations to convert string 1 to string 2 is: 0
The runtime to convert string 1 into string 2 is: 7648894 nanoseconds.
```

One last example shows that the more errors to be changed/optimizations leads to higher runtime. Here are strings of length 100 with high error count which is less than the same length with low error count:

```
First string to be converted:
AACCAAGACCTATGCGACTAGCATCCTAATCACCACTAGTAAAGTAATACGCATCTACCGGCTGTACGAATCTTGTA
CACGATTAGTGAAAAAGGC
Second string for first string to be converted to:
ATCCACACCTTTCCTTTAGTCCTACTGCGAACCTATCTAGGAAACAGCAGACACAGTCTGTATAGACATTGCGTTCA
TAGAACAGCGTT
The number of operations to convert string 1 to string 2 is: 44
```

```
The runtime to convert string 1 into string 2 is: 85394 nanoseconds.

First string to be converted:
AACCAAGACCTATGCGACTAGCATCCTAATCACCACTAGTAAAGTAATACGCATCTACCGGCTGTACGAATCTTGTA
CACGATTAGTGAAAAAGGC
Second string for first string to be converted to:
TCTAGGGACCCTTTCTCGAGTACATCGAAACCACCGTTTTATGAAGCTGCACATCTCCGTGATTATGTTATTTTTAC
TTCTTCTCATGGTGAAGACCTC
The number of operations to convert string 1 to string 2 is: 47
The runtime to convert string 1 into string 2 is: 95657 nanoseconds.
```

Conclusions

Something that I learned this assignment is how intuitive dynamic programming is. I always get confused on how to implement recursive functions, as the recurrences always confused me, but dynamic programming solves this issue and can sometimes result in a better solution than a recursive one. I also learned how easy it is to translate dynamic programming algorithm concepts to code, as this assignment was relatively easy to code. The concepts are directly translatable into code just by looking at the concept of the algorithm, while if I had no prior insight to these algorithms, I don't even know where I would've started! So, overall, this assignment gave me a huge appreciation for using dynamic programming as a method in which to solve problems with algorithms.