Project 2

Mengxiang Jiang
CSEN 5336 Analysis of Algorithms

October 23, 2022

Problem 1. Two DNA strands of length 100 (number of bases in the sequence) are given in DNA1.txt and DNA2.txt file.

a) Find the longest common subsequence (LCS) between the two sequences using dynamic programming bottom-up (tabulation) approach. Submit your code to solve the problem. How much time (in seconds or milliseconds) is required by your computer to run the algorithm?

```
def lcs_bottom_up(s1, s2):
    c = [[0]*len(s2) for i in range(len(s1))]
    s = [['']*len(s2) for i in range(len(s1))]

    for i in range(1, len(s1)):
        for j in range(1, len(s2)):
            if s1[i] == s2[j]:
                c[i][j] = c[i-1][j-1] + 1
                s[i][j] = s[i-1][j-1] + s1[i]
            else:
                if c[i][j-1] > c[i-1][j]:
                    c[i][j] = c[i][j-1]
                    s[i][j] = s[i][j-1]
                else:
                    c[i][j] = c[i-1][j]
                    s[i][j] = s[i-1][j]
    return s[-1][-1]
```

Listing 1: Find LCS Bottom Up

b) Solve the same LCS problem using dynamic programming top-down (memorization) approach. Submit your code to solve the problem. How much time (in seconds or milliseconds) is required by your computer to run the algorithm.

```
def lcs_top_down(s1, s2):
    memo = [[-1]*len(s2) for i in range(len(s1))]
    result = [['']*len(s2) for i in range(len(s1))]
    def lcs(i, j):
        if i == 0:
            return 0
        if j == 0:
            return 0
        if memo[i][j] != -1:
            return memo[i][j]
        if s1[i] == s2[j]:
            memo[i][j] = 1 + lcs(i-1, j-1)
            result[i][j] = result[i-1][j-1] + s1[i]
            return memo[i][j]
        else:
            a = lcs(i, j-1)
            b = lcs(i-1, j)
            if a > b:
                memo[i][j] = a
                result[i][j] = result[i][j-1]
            else:
                memo[i][j] = b
                result[i][j] = result[i-1][j]
            return memo[i][j]
    lcs(len(s1)-1, len(s2)-1)
    return result[-1][-1]
```

Listing 2: Find LCS Top Down

c) Which solution is faster and why? Discuss your solutions

```
if __name__ == '__main__':
    s1 = ''
    s2 = ''
    with open('DNA1.txt', 'r') as f:
        s1 = f.read().rstrip()
    with open('DNA2.txt', 'r') as f:
        s2 = f.read().rstrip()
    import time
    bottom_up_start = time.perf_counter()
    bottom_up_result = lcs_bottom_up(s1, s2)
    bottom_up_end = time.perf_counter()
    print("Bottom up took %f seconds and result is %s"
    % (bottom_up_end-bottom_up_start, bottom_up_result))
    top_down_start = time.perf_counter()
    top_down_result = lcs_top_down(s1, s2)
    top_down_end = time.perf_counter()
    print("Top down took %f seconds and result is %s"
    % (top_down_end-top_down_start, top_down_result))
```

Listing 3: main function code

```
$ python3 lcs.py
Bottom up took 0.002600 seconds and result is
    TCTTTTTCTTTGTTTGGCACCGTTGCGGCTACCTTGCAGCTCTGCATCTTGGCTCTACTGCACT
Top down took 0.003055 seconds and result is
    TCTTTTTCTTTGTTTGGCACCGTTGCGGCTACCTTGCAGCTCTGCATCTTGGCTCTACTGCACT
```

Listing 4: Terminal output running code

The bottom up solution is about 15% faster than the top down solution. This is probably due to not requiring context switching and a recursive stack. The filling out the table is also more straightforward compared to the memoization, which could also contribute to the speed.