CS 341 A3 Q1

Tareef Dedhar - 20621325

June 16, 2018

1 LCS of N Strings

1.1 Description/Correctness

For each string we will keep track of our "current position" as an element of the array positions. This is initialized with the max length of each string as the value for its index. Then, we call LCS on this array. If the character at the position is equal for all strings, we add 1 to the solution, decrement all indices, and recurse. If not, then we take the max of each possible sub-problem where these are recursing while decrementing only one position (so n recursive calls). This solution can and should use caching of calculated values, of course (this can be implemented as an n-dimensional array, where each possible LCS value for each state of positions[] is stored once calculated). This algorithm considers all options, as if all strings end with the same value that must be part of the subsequence, and if not, this considers the subproblem for each string as being the one to decrement, and follows the same logic in each recursive call, effectively checking every possible combination of strings.

1.2 Pseudocode

```
for strings 0 thru n:
int positions[n] = length of each string at the given index

LCS(int positions[]):
if (s0[positions[0]] == s1[positions[1]] == ... == sn[positions[n]]):
  for each i in positions:
    positions[i] -= 1
  return 1 + LCS(positions)
else:
  int temp[n]
  for each i in temp:
    temp_positions = positions
    temp_positions[i] -= 1
    temp[i] = LCS(temp_positions)
  return Max(temp)
```

1.3 Time Complexity

The time complexity of this algorithm is as follows: Generating the inital array of positions, assuming length is a known quantity of each string, takes Θ(n) time (where n is the amount of strings). From there, we consider the initial call to the recursive function. We first iterate through the array, checking for equality. This takes Θ(n) time. If this check is true, we recurse on the same set of strings with 1 less character. However, this is the best case. In the worst case, we perform this same calculation and check, but recurse n times, with only 1 out of the n strings being decremented. If there is no common subsequence, then this means we have n calls which run n calls themelves, which means we have a runtime of O(n^n).