The University of West Florida
Department of Computer Science
Data Structures and Algorithms II
John W. Coffey

Background

As discussed in class, the longest common subsequence (LCS) problem is a common problem that arises (e.g.) in computational biology. Given two sequences of numbers, letters, DNA sequences, etc., `x[1..m]` and `y[1..n],` find a longest subsequence that is common to both of them. Usually no unique answer, but rather several possible answers exist. As we know, the brute force algorithm is O ($n*2^m$) whereas the dynamic programming version is O(m*n). However, as we also know, the basic dynamic programming solution requires O(2*n*m) space.

Program Descriptions

Assume that we are implementing a program to help us study similarity among organisms. In order to do this, we will implement the dynamic programming version of the LCS algorithm in two different ways and in the same program.

Part 1:
The first part of the program will read a file containing a pair of character strings ("`twoSequences.txt`") corresponding to the sequences, compute the LCS, display the original strings and the longest common subsequence of characters. This version of the problem will utilize the entire "c" array, but NO "b" array. You must reconstruct the sequence itself from the "c" array for full credit. If you use the "b" array, it will cost you 10%.

Part 2:
The second part will not make any attempt to recount the actual subsequence. It will calculate measures of similarity (which I made up) among an arbitrary number of character strings in a file named "`multipleSequences.txt`". This file will start with an integer (on its own line in the file) indicating the number of strings that follow. That number of strings will follow, one per line. This second part of the program will utilize an approach of keeping only the `2xM` entries needed to compute the maximum value of EACH LCS.

It will produce as output, a table that will look like this:

```
    1  2  3  4  5  6  7
 1  -  H  M  D  M  L  D
 2  -  -  H  H  D  D  H
 3  -  -  -  L  M  D  M
 4  -  -  -  -  M  L  M
 5  -  -  -  -  -  M  L
 6  -  -  -  -  -  -  M
 7  -  -  -  -  -  -  -
```

Obviously, the 1, 2, 3, ... are labels for the strings, and the upper triangle holds a measure of similarity between pairs of strings {1,2}, {1,3}, {1,4}...

The possible entries in the table are:

```
H = high similarity between the strings
M = medium similarity between the strings
L = low similarity between the strings
D = the two strings are dissimilar
```

Determining similarity:

- High similarity exists if the length of the shorter string is within 10% of the length of the longer string and the longest common subsequence is at least 80% of the length of the shorter string.
- Medium similarity exists if the criteria for High similarity is not met but the length of the shorter string is within 20% of the longer string and the longest common subsequence is 60% of the length of the shorter string.
- Low similarity exists if the criteria for Medium similarity is not met but the length of the shorter string is within 40% of the longer string and the longest common subsequence is 50% of the length of the shorter string.
- Dissimilar strings are any that meet none of the above criteria.

The file of strings will first contain an integer that indicates how many strings are in the file, followed by that number of character strings. Strings will be terminated in the file by newlines. Recognize that you can end up with memory problems and you should not read all the strings into RAM at once. This is an obvious place to use a direct access file. We are already economizing on the amount of ram by not using the O(2*m*n) space. The reason to take this approach is that these will be long strings and there might be a lot of them.

Deliverables
You will submit the following for this project:
1. Your functional decomposition.
2. A User's manual for your program
3. Your test versions of the `twoSequences.txt` and `multipleSequences.txt` files
4. Your source code in C.

Please review the policy on Academic dishonesty. This is an individual project.

Submission:
1. Compile and run your program one last time before submitting it. Your program must run with gcc in the Linux lab.
2. Place every file in your submission in a SINGLE DIRECTORY named <first initial><last name>-p<number>. For instance, I would create directory:
    jcoffey-p1 for project 1.
3. zip that FOLDER into a .zip file with the SAME NAME. This means that inside your zip file, you will have exactly one folder (from the example: jcoffey-p1) showing at the top level. Inside that folder will be ALL the files in your project.
4. DO NOT make separate folders for documentation and source files (or anything else) inside the main folder. Having such a setup simply necessitates more navigation time to get where we need to go to grade.
5. Any needed input files should be in the top-level folder along with the source code.
6. MacOS users - remove the _MacOS_ utility folder before you zip up the file. If you cannot, delete it from the archive once the archive is created. It just takes up space and is not needed for anything we

do with your submissions.
7. Login to UWF's eLearning system at `http://elearning.uwf.edu/`. Select our course.
9. Select the Dropbox option. Then select the appropriate project folder.
10. Upload your zip file to that folder. Check the dropbox to insure that the file was uploaded.
ALWAYS give yourself enough time. If you are trying to submit at 11:57pm on your machine, the clock might be off and the dropbox might already be closed.

Please review the policy on ACADEMIC MISCONDUCT. This is an *individual assignment.*