Data Structures and Algorithms (CS 130b) **Prof. Suri**

Programming Assignment 2

Handed Out: Feb 10 *Due: Feb 26, 11:59 PM*

Your second programming assignment asks you to implement the **Longest Common Subsequence** algorithm. Your program should read two character strings, and it should output (1) the **length** of the LCS, and (2) the common subsequence as a character string.

Recall that the dynamic program to determine the LCS length is the following:

LCS-Length (X, Y)

```
for i = 1 to m do c[i, 0] ← 0;
for j = 1 to n do c[0, j] ← 0;
for i = 1 to m do
   for j = 1 to n do
      if x_i = y_j then
         c[i, j] ← c[i − 1, j − 1] + 1
         b[i, j] ← D
      else if c[i − 1, j] ≥ c[i, j − 1] then
         c[i, j] ← c[i − 1, j]
         b[i, j] ← U
      else
         c[i, j] ← c[i, j − 1]
         b[i, j] ← L
return b, c
```

The pseudo code for producing the actual subsequence of maximum length is given below. It uses the b array computed above.

PRINT-LCS(b, X, i, j)

```
if i = 0 or j = 0 then return
if b[i, j] = D then
   PRINT-LCS(b, X, i − 1, j − 1)
   print x_i
elseif b[i, j] = U then
   PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

Makefile [5 pts]

You will need a makefile once again. Be sure to follow the specification—*this is an easy 5 points!* The default target (the first one in the makefile) should be called `all` and should compile the appropriate source files. More precisely, running `make all` should create an executable file named `findLCS`. The `clean` target should cleanup all output files generated by running `make all`. Note that `clean` should not complain if there's nothing to do and should not require any interaction, so make sure to use "`rm -f`" instead of just `rm`.

We will accept any programming language as long as the executable `findLCS` accepts the desired command-line arguments and produces expected results. For example, if you are using *java*, you could write a wrapper script and call it `findLCS`. This wrapper script will then simply run your java class with the command-line args that were passed to the script.

Your program `findLCS` will have two components as discussed below.

LCS [75 pts]

The input for this part will have the following format.

- The first line contains a number N that indicates the number of testcases to follow.
- Each test case is a single line that contains exactly two strings separated by a whitespace. These strings contain ASCII coded alpha-numeric characters.

Following is an input file `small.txt`:

```
3
abcdbcadcda abbcddab
abab cab
1021 210
```

For each of these testcases, your program should print out (1) the length of longest common subsequence of the two input sequences, and (2) the LCS itself as a string, followed by a newline.

For instance, on input '`abcdbcadcda`' '`abbcddab`', your program should output '`7`' and '`abbcdda`'. If there are more than one LCS possible, just report one of them: for instance, on input '`1021`' and '`210`', output '`2`' as length, and either '`10`' or '`21`' as LCS.

An example of how the program will be run and its expected output is the following.

```
user@csil:~] ./findLCS < small.txt
7 abbcdda
2 ab
2 10
```

(Note that for the last example, either '`2 10`' or '`2 21`' is a valid output.)

All LCS [20 pts]

In this part, we want to report all the longest common subsequences for a pair of strings. Note that this can produce identical strings due to character matches occurring at different locations in the sequence. Therefore, you must produce the output in a different form. Now, instead of writing out the LCS string, you must write out a *mapping* of the *index locations* in the input sequences where the characters of the LCS match.

For example, if the input is $A =$ '`abab`' and $B =$ '`cab`', then the LCS string is $L =$ '`ab`' but there are three distinct mappings possible. Each of these mappings will be represented by a unique tuple of the following form

$$(< I(A, L_1),\ I(B, L_1) >,\ \ \ldots,\ \ < I(A, L_k),\ I(B, L_k) >),$$

where $I(A, L_i)$ is the index of the character L_i in string A and k is the size of the LCS.

For instance, on input $A =$ '`abab`' and $B =$ '`cab`', the outputs will be `(<3,2>, <4,3>)`, `(<1,2>, <2,3>)`, and `(<1,2>, <4,3>)`. The first tuple corresponds to the LCS $L =$ '`ab`' with a mapping $A_3 \rightarrow B_2$ for '`a`', and $A_4 \rightarrow B_3$ for '`b`'. The next tuple corresponds to $A_1 \rightarrow B_2, A_2 \rightarrow B_3$, and similarly the third tuple corresponds to the mapping $A_1 \rightarrow B_2, A_4 \rightarrow B_3$.

Your program should accept a command line argument '`-all`' to turn on processing for this part. The input will be the same as the previous part. For each test case (that is a pair of strings A and B), your program should output one mapping per line. Once all the possible mappings for A and B have been printed, output a blank line and then proceed to process the next input in the test case.

An example of ALL-LCS program run and its expected output:

```
user@csil:~] ./findLCS -all < small.txt
(<1, 1>, <2, 2>, <5, 3>, <6, 4>, <8, 5>, <10, 6>, <11, 7>)

(<3, 2>, <4, 3>)
(<1, 2>, <4, 3>)
(<1, 2>, <2, 3>)

(<1, 2>, <2, 3>)
(<3, 1>, <4, 2>)
```

The first line is a mapping of index locations for the LCS '`abbcdda`' of the strings '`abcdbcadcda`' and '`abbcddab`'. The next three non-empty lines are the three possible mappings for the LCS '`ab`' of the strings '`abab`' and '`cab`'. Finally, the last two non-empty lines are the mappings for the two longest common subsequences '`10`' and '`21`' respectively of the strings '`1021`' and '`210`'.

Note: Make sure that the tuples are sorted in increasing order of the index in the first string. That is, `(<1, 2>, <2, 3>)` is valid but `(<2, 3>, <1, 2>)` is not. However, any vertical ordering of these tuples is acceptable.

Grading

Your programs will be graded automatically, so make sure that the output produced by your program strictly adheres to the prescribed format. One easy way to ensure this is to check if your output for the input file '`small.txt`' exactly matches the expected outputs shown before.

Your programs will be graded as per the following marking scheme.

- *Small Inputs : 20%*

  We will run your program over an input file with $N \leq 10$ testcases and the size of strings A and B less than equal to 10.

- *Medium Inputs : 30%*

  We will run your program over an input file with $N \leq 100$ testcases and the size of strings A and B less than equal to 100.

- *Large Inputs : 50%*

  We will run your program over an input file with $N \leq 100$ testcases and the size of strings A and B less than equal to 1000.

Turnin Instructions

In order to electronically turnin your project files, use the following command:

```
turnin prog2@cs130b LIST
```

where LIST is the list of all your files, i.e., your makefile, all your source and header files. (The turnin program is located in `/usr/bin/turnin` on `csil.cs.ucsb.edu` only.) If you turnin more than once then the version that will be graded is the last version you turned in before the deadline.