**Project 5: String Matching**

***Longest Common Subsequence and Edit Distance***

**Version 05/18/19**

**Educational Objectives:** After completing this assignment, the student should be able to accomplish the following:

* State in detail the Longest Common Subsequence problem (LCS).
* State and implement algorithms for solving the LCS.
* State in detail the Edit Distance problem (ED).
* State and implement algorithms for solving the ED.
* Describe and give examples of applications of LCS and ED in computational biology.

**Operational Objectives:** Work Exercises 1,2,3 in Course Notes: [Strings 4](http://www.cs.fsu.edu/~lacher/courses/notes/strings4.html).

**Deliverables:**

LCS.cpp # implements Longest Common Subsequence [solution to Exercise 1]

ED.cpp # implements Edit Distance [solution to Exercise 3]

log.txt # work log / test diary [begins with solution to Exercise 2]

**Procedural Requirements**

1. The official development/testing/assessment environment is specified in the Course Organizer. Code should compile without warnings or errors.
2. In order not to confuse the submit system, create and work within a separate subdirectory cop4531/proj5.
3. Maintain your work log in the text file log.txt as documentation of effort, testing results, and development history. This file may also be used to report on any relevant issues encountered during project development.
4. **General requirement on genericity.** Use generics whenever possible. For example:
   1. If you need to sort, use a generic sort algorithm or a container with built-in sorting.
   2. If a sort requires a specialized notion of order, create a function object that captures the desired order property.
   3. If you need a graph algorithm, use a component of the fsu graph library.
   4. In general, whenever a generic algorithm exists that can be deployed, do not circumvent that with specialized one-off code.
   5. Carefully choose all containers as the most appropriate for a particular purpose.

In short: re-use components from LIB (or your own versions) whenever possible. Don't re-invent the wheels.

1. Begin by copying these files into your project directory:
2. LIB/proj5/deliverables.sh # submission configuration file
3. LIB/area51/LCS\_i.x # benchmark Longest Common Subsequence executable
4. LIB/area51/ED\_i.x # benchmark Edit Distance executable

Note that the benchmark executables may exhibit different behavior than the samples in LIB/notes\_support of the same name.

1. Create the files LCS.cpp and ED.cpp as described in Exercises 1 and 3.
2. Create the file log.txt as usual, and add solutions to Exercise 2.
3. Follow these steps when you are ready to submit:
   1. Be sure that you have established the submit script LIB/scripts/submit.sh as a command in your ~/.bin directory. [We are currently using version 2.0.]
   2. Copy the submit configuration script LIB/proj5/deliverables.sh into your project directory. [This is an important step, as deliverables may have changed.]
   3. Submit the project using the command submit.sh while in your project directory and logged in to shell or quake. Pay attention to the screen, which will warn you about missing deliverables and incorrect login.
   4. Check your CS email for feedback. If Robocheck is active for this project, read the results from automated testing.

***Warning:****Submit scripts do not work on the program and linprog servers. Use shell.cs.fsu.edu or quake.cs.fsu.edu to submit projects. If you do not receive the second confirmation with the contents of your project, there has been a malfunction.*

**Code Requirements and Specifications**

1. These libraries may NOT be used:
2. <string>
3. <set>
4. <unordered\_set>
5. <map>
6. <unordered\_map>
7. <algorithm>

There are equivalent components of the fsu::library in ~cop4531p/LIB that may be used.

1. Be sure that the source code files LCS.cpp and ED.cpp compile directly to executables using the c4531 command line compile macro:
2. c4531 LCS # produces LCS.x
3. c4531 ED # produces ED.x
4. Don't make the mistake of assuming standards are down because we are allowing one-command compile. At this stage of your career, all code that you write should be up to high standards of design, formatting, wise name choices, self-documenting code and design, and all the other things that make code pleasurably readable.

In particular, we recommend using two enumerated types DIR = { ZERO, LEFT, UP, DIAG }; (for direction in the P matrix) and EDITS = { Delete, Insert, Substitute, Match }; (for edit transcripts) with overloaded output operator<< for each type. This alone will make your code more readable and maintainable.

No global variables, and large objects (e.g., sequences and matrices) should be passed by reference or pointer (when the function call is expected to modify the object) or const reference or pointer (when no modification should occur).

1. LCS.x and ED.x should exhibit behavior identical to their LIB/area51 benchmarks.

**Hints**

* Check out:
* LIB/tcpp/matrix.h # matrices
* LIB/tcpp/genalg.h # fsu::Max
* Note that fsu::Matrix < T > has some useful methods (aside from the "double" bracket operator[][]) - particularly Dump(std::ostream& os , size\_t width).
* When taking the Min of three things, we suggest coding it directly using this logic:
* // Min (A,B,C)
* val = A;
* if (B < val) val = B;
* if (C < val) val = C;

This means that A wins ties with either B or C and B wins ties with C. This will be helpful in getting consistent results in the parent matrix P. Using operators:

Min (A,B,C) = (A <= (x = (B <= C ? B : C)) ? A : x); // <-- this

Min (A,B,C) = (A < (x = (B < C ? B : C)) ? A : x); // <-- NOT this