CS4040/5040
Homework # 5

due by midnight Friday, November 20, 2020

Program 1 The motivation for this problem is the issue of determining the similarity between two sequences of DNA. DNA sequences can be extremely long, for example, human DNA is billions of units long. Determining the similarity between two strings of DNA therefore must use very efficient algorithms to compare the DNA. For our problem, we will compute a similarity measure between "genes" of DNA, not on the actual base pairs themselves. So we will need a way to determine what part of a DNA sequence denotes a particular gene, and then whether that gene is similar to a gene in a second DNA sequence. Note: appologies to any biochemists as I know this is not exactly how DNA works, I've simplified it for the purposes of this assignment.

In this problem we will abstract the above notions, and try to make the problem tracktable, meaning that an efficient solution is possible.

We seek to compute the longest common gene subsequence between 2 input strings. The strings can be VERY long. Instead of considering the strings as sequences of characters, we will look at them as sequences of n/k genes of length k, with n always equally divided by k. In this problem, two length k genes are deemed to match if they contain the same ASCII characters in the same quantity, but possibly in a different order. The goal of this assignment is to develop an extremely efficient algorithm to determine the longest common subsequence of matched genes. Any valid ASCII character can be a component of a gene. So genes can be any length k ASCII string.

For example (a very small example, just to illustrate the concept):

```
k: 4
String 1: "ABCDACBDAAAA"
String 2: "BADCACBB"
```

So string 1's first "gene" is "ABCD", while string 2's first gene is "BADC". These two gene's match as they contain the same number of each character. The second "gene" in each are "ACBD", and "ACBB". These genes will not match. The longest common gene subsequence between these strings would therefore be 1 gene long, "ABCD".

Example 2:

```
k: 3
String 1: "zwawazfee"
String 2: "wazzaaeef"
```

In this case, the longest common gene subsequence would be "zwa**fee**" or two genes in length.

Example 3:

```
k: 3
String 1: "zwawazfee"
String 2: "wazzaaeff"
```

In this case, the longest common gene subsequence would be “zwa” or one gene in length, as “fee” does not match “eff”.

Note: the test strings we will use will be *MUCH* longer. Also, k may be any length, from very small like in the above examples to *MUCH* larger, possibly millions of characters long.

So that the grader and I can have a common interface to your code, the input to the program will be in a file whose first line is k, second line will be string 1, and third line will be string 2. Note: as we are using lines delimited by a newline character in the file, that and only that ASCII character will not be used in either string 1 nor string 2. We will execute your code by typing in the following to the terminal:

```
% ./prog1 <testfilename.txt>
```

Your program should return first the length of the longest common gene subsequence found on the first line of output, followed by the subsequence itself on the second line.

For example, if the following was in the file `test.txt`:

```
3
zwawazfee
wazzaaeef
```

And I typed the following in the terminal:

```
% ./prog1 test1.txt
```

Your program should output:

```
2
zwafee
```

Your program will be graded on efficiency, as well as style and correctness. You must analyze the efficiency of all the algorithms used in your program and include this analysis in comments in your code. In particular, you **must** document the efficiency of *each* function in your code with an asymptotic analysis for both time and space. You **must** also do this for any code you might include from other sources. In addition to the in-line comments, you **must** submit an overall analysis of the entire program, detailing the top-level tasks, and datastructures, with justifications and explanations of any design decisions you make in the file `analysis.txt` and submitted with the rest of your program.

If you have any questions about how to implement this program, please email me.

Your programs must conform to the style guidelines as given on the course homepage.

To submit your assignment, upload a compressed tar (.tar.gz, or .tgz) version of your code in a directory with your user name and documentation via blackboard. If

you are unfamiliar with the tar utility, please take a look at:
`https://www.thegeekstuff.com/2010/04/unix-tar-command-examples/`
Note: do not submit extraneous files, nor executables, only source code and documentation which should all be in a single directory with no subdirectories called "YourFirstNameYourLastName". Obviously, you should replace that text with your own first and last names!

Be sure to submit a makefile that compiles the code in your directory. I should be able to compile the code in the directory by typing `make`. The result of typing `make` must be creation of an executable called `prog1`. If it doesn't compile when I do this, your code is not in compliance with the requirements for this assignment, and you will get a 0 for this assignment. The code will be tested on the Ubuntu machines in Stocker.

To test your code, you can upload your code to your account on these machines (if you don't have an account or have forgotten your password, email our sysadmin `Carl Hawes <hawes@ohio.edu>` for help), and run it remotely via the ssh/scp protocols. For more detail on these see:
`http://oucsace.cs.ohio.edu/~chelberg/classes/2400/remote_login.html`