CS325 programming assignment 2

Electronic submission of Code and Report to TEACH
at 11:59PM Feb 23rd

February 8, 2017

TAs in charge Souti Chattopadhyay (chattops@oregonstate.edu) Hamed Shahbazi(shahbazh@oregonstate.edu)

Sequence alignment via dynamic programming

One of the central problems pursued in bioinformatics is determining similarity between sequences of nucleotides or amino acids (the basic information that makes up our genetic structure). Similarity information is required to perform some typical tasks including: using a given sequence of amino acids to infer a proteins shape and function, and finding all the genes and proteins in a given genome.

In this assignment, you will solve the problem of aligning two gene sequences optimally using dynamic programming. Given two gene sequences, our goal for global alignment is to find the optimal alignment of the two sequences to minimize the alignment cost, which is defined by a symmetric cost matrix like the one below:

	-	A	T	G	C
-	0	1	1	1	1
A	1	0	2	2	2
T	1	2	0	2	2
G	1	2	2	0	2
C	1	2	2	2	0

This cost matrix assign a cost of 0 if two identical characters are aligned, a cost of 1 if we align a character with a space (deletion or insertion), and a cost of 2 if two different characters are aligned (substitutions). The following alignment of two sequences (ATCC and TCAC) have a total cost of 6.

sequence 1:	A	-	T	C	C
sequence 2:	T	C	A	-	C
Cost:	2	1	2	1	0

A better alignment is as follows with a total cost of 2.

sequence 1:	A	T	C	-	C
sequence 2:	-	T	C	A	C
Cost:	1	0	0	1	0

Note that this can be viewed as a generalization of the edit distance problem introduced in class, whose cost matrix has a cost of 1 for all insertions and deletions and substitutions. You should adapt the dynamic programming solution presented in class for edit distance to solve this generalized alignment problem.

1 Specification of the input and output

Please follow below instructions before starting to develop your program. Your program will take two input files. The first specify the cost matrix. The second file specify sequences to be aligned. Below we describe the format of these two files.

Input file 1: Cost File The cost will be specified by a 6 by 6 matrix like the one shown above. Each row of the matrix will be one line. The first row and the first column will specify the alphabet. The remaining entries specifies the cost of aligning a particular pair of alphabets. Please see **imp2cost.txt** as an example cost file in your program. During grading, we use cost files with the same name and structure as **imp2cost.txt**. We will only change the values but the alphabets will be the same and the matrix will remain symmetric.

Input file 2: Input sequences Each sequence input file can contain multiple pairs to be aligned. Each line of the file contains one pair of sequences to be aligned. The two sequences are separated by a single comma. We have provided some sample input files named **imp2input.txt**.

Output file Given the inputs, the output of your program will be a single txt file. Given an input sequence file contains multiple pairs, the output file will also contain multiple lines, each line reporting the result for one pair. In particular, the line will begin with the alignment of the two sequences, separated by comma, then followed by the cost of the alignment. For example, if your input is "AATC" and "ATC", the alignment output will be "AATC, A-TC:1", assuming cost of 1 for deletion. For the given input file, we have also provided an output file **imp2output_our.txt**.

2 Empirical Verification of correctness.

The TAs will run your algorithm on their inputs and verify the correctness of the algorithm considering two aspects. First, we will verify if your program arrives at the correct optimal cost for the given inputs. This will be achieved by comparing the cost you report with the optimal cost. We will further verify if the alignment your output is correct. This will be done by checking if the cost of

your reported alignment is actually the same as your reported cost. Specifically, we will use a python script, which take the cost file and the output file your algorithm produced, and verify if the reported alignments and reported costs match up. To help you debug your program and run the verification your self, the script will be provided. To run the script, simply use:
python coster.py -i imp2output.txt -c imp2cost.txt -o imp2out_c.txt

The result of the verification will specify for each line whether it passes or fails the verification.

Our Test Expectation During testing, we will place your program into one main folder containing our own inputs
imp2input.txt, **imp2cost.txt**. Since we will not pass any file names as parameters to your program, please make sure that your program works with the default names **imp2input.txt**, **imp2cost.txt** and is able to find them in the same folder. Finally, your output file should be placed the same folder under name
imp2output.txt.

3 Empirical analysis of run time.

For this part, you need to generate your own random inputs of sequences of five different lengths: 500, 1000, 2000, 4000, 5000. Feel free to consider even longer inputs to push the limit your implementation. For each length, please generate 10 random pairs of sequences (using alphabet A, G, T and C) and compute the pairwise alignments and report the average time for computing the alignments.

4 Report.

In addition to submitting the source code (with clear instructions for running them), you will also need to submit a report, which should include the following:

- **Pseudo-code.** Please provide the pseudo-code for your algorithm.
- **Asymptotic Analysis of run time.** Please analyze the runtime for your algorithm.
- **Reporting and plotting the runtime.** Describe how the running time is measured in your experiments. What type of machine is used for the experiments. Is there any part excluded from the runtime measurements (e.g., output of the alignment)? Plot the running times as a function of the input size. Label your plot (axes, title, etc.). Include an additional plot of the running times on a log-log axis [1]. Note that if the slope of a line in a log-log plot is m, then the line is of the form $O(x^m)$ on a linear plot. Determine the slope of the line in your log-log plot and from the slope, infer the experimental running time for your algorithm.
- **Interpretation and discussion** Discuss the runtime plot. Do the growth curve match your expectation based on their theoretical bounds?

[1] See here for a tutorial on log-log plots: `http://www.eecs.orst.edu/~glencora/cs325/videos/loglogplots.mp4` (and accompanying file: `https://web.engr.oregonstate.edu/~glencora/wiki/uploads/loglogplots.m`).

5 Hand in Instructions

Code and report should be submitted through the TEACH website. Students are encouraged to work in groups of up to 3 people. Each group **should submit only one submission**, which should include the names of all group members. You can choose any of the following programming languages: `java, python, c/c++/c#`. If you wish to use a language that is outside of this list, you will need to obtain advance approval from at least one of the TAs in charge.