CS325 implementation assignment 2: Sequence alignment via dynamic programming

A central problem pursued in bio-informatics 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 protein's 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 particular cost matrix assigns 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 cost 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.

Note: We will check the costs and not the sequences as multiple solutions are possible with the same cost depending on how you have resolved ties.
Your code will need to compiled and run on flip. The TA 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, you can use the provided `checker.py` with the following command:

```
python check_cost.py -c <costfilename> -o <outputfilename>
```

If no filenames are provided, it will assume a default name for cost file `imp2cost.txt` and default name for output file `imp2output.txt`. If there is any mismatch between the expected cost and reported cost, the program will detect and report them in its result file "cost_check_results.txt".

Your can further verify if your output correctly matches the optimal cost specified by the provided solution file with the following command:

```
python check_cost.py -c <costfilename> -o <outputfilename> -s <solutionfilename>
```

If `solutionfilename` is not provided, it will only check if the provided cost matches the provided alignment in the outputfile.

TA Test Expectation During testing, we will place your program into a folder containing our own inputs **imp2input.txt**, **imp2cost.txt**. To simply the testing, we will not pass any file names as arguments to your program, please ensure that your program works with default inputs **imp2input.txt** and **imp2cost.txt** in the same folder and produces a output file in the same folder with the default name `imp2output.txt`. We will test this output for formatting issues and compare it against our own solution file to see if your program finds the optimal alignments for the given inputs.

Checklist for code expectation:

- `python <your_script_name>.py` should generate `imp2output.txt` when placed in the same folder with `imp2input.txt` and `imp2cost.txt`.
- `python check_cost.py` passes successfully, indicating no formatting error and the reported costs are consistent with the provided alignments.

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. See details in the Report section for expected format for reporting.

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 empirical 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.). Please use log-scale for both x and y axes of your plot. Note that if the runtime is of the form $O(x^m)$, the log-log plot will be a line with the slope equals to m. 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—does the growth curve match your expectation based on the asymptotic analysis?