CPE 349: Edit Distance/Global Sequence Alignment

Write a program to compute the optimal sequence alignment of two DNA strings. This program will introduce you to the emerging field of *computational biology* in which computers are used to do research on biological systems.

Biology review. A *genetic sequence* is a string formed from a four-letter alphabet {Adenine (A), Thymine (T), Guanine (G), Cytosine (C)} of biological macromolecules referred to together as the **DNA bases**. A *gene* is a genetic sequence that contains the information needed to construct a protein. All of your genes taken together are referred to as the human genome, a blueprint for the parts needed to construct the proteins that form your cells. Each new cell produced by your body receives a copy of the genome. This copying process, as well as natural wear and tear, introduces a small number of changes into the sequences of many genes. Among the most common changes are the substitution of one base for another and the deletion of a substring of bases; such changes are generally referred to as *point mutations*. As a result of these point mutations, the same gene sequenced from closely related organisms will have slight differences.

The problem. Through your research you have found the following sequence of a gene in a previously unstudied organism. `A A C A G T T A C C` **What is the function of the protein that this gene encodes?** You could begin a series of experiments in the lab to determine what role this gene plays. However, there is a good chance that it is a variant of a known gene in a previously studied organism. Since biologists and computer scientists have laboriously determined (and published) the genetic sequence of many organisms (including humans), you would like to leverage this information to your advantage. We'll compare the above genetic sequence with one which has already been sequenced and whose function is well understood. Say `T A A G G T C A` If the two genetic sequences are similar enough, we might expect them to have similar functions. We would like a way to quantify "similar enough."

Edit-distance. In this assignment we will measure the similarity of two genetic sequences by their *edit distance*, a concept first introduced in the context of coding theory, but which is now widely used in spell checking, speech recognition, plagiarism detection, file revisioning, and computational linguistics. We align the two sequences, but we are permitted to *insert gaps* in either sequence (e.g., to align characters later in the sequence). We pay a penalty for each gap that we insert and also for each pair of characters that mismatch in the final alignment. Intuitively, these penalties model the relative likeliness of point mutations arising from deletion/insertion and substitution. We produce a numerical score according to the following table, which is widely used in biological applications:

operation	cost
insert a gap	2
align two characters that mismatch	1
align two characters that match	0

Here are two possible alignments of the strings *x* = `"AACAGTTACC"` and *y* = `"TAAGGTCA"`:

x	y	cost
A	T	1
A	A	0
C	A	1
A	G	1
G	G	0
T	T	0
T	C	1
A	A	0
C	-	2
C	-	2

		8

x	y	cost
A	T	1
A	A	0
C	-	2
A	A	0
G	G	0
T	G	1
T	T	0
A	-	2
C	C	0
C	A	1

		7

The first alignment has a score of 8, while the second one has a score of 7. The *edit-distance* is the score of the best possible alignment between the two genetic sequences over all possible alignments. In this example, the

second alignment is optimal, so the edit-distance between the two strings is 7. Computing the edit-distance is a nontrivial computational problem because there are **exponentially many possibilities**.

A direct implementation of a recursive scheme will work, but it is spectacularly inefficient. To overcome this, use a *dynamic programming approach* to both compute the edit distance and then recover the alignment that corresponds to the edit distance. (In the case of multiple alignments resulting in the same edit distance, any one of the alignments can be reported.) To recover the alignment **retrace** the steps of the dynamic programming algorithm backwards, **re-discovering** the path of choices. **Do not use an additional data structure to track the program's choices as you compute the edit distances.** You create and use only the data structure that stores the edit distances of the subproblems used to compute the overall edit distance. When back tracing you may use an additional data structure if it is convenient. (You must implement back tracing!)

Your program. Write a program `EditDistance.java` that reads two strings of characters from a text file. (Although, in the application described, the characters represent genetic sequences, your program should handle any sequence of alphanumeric characters.) Your program should then compute and print the edit distance between the two strings. Finally, it should recover the optimal alignment and print it out along with the individual penalties, using the following format:

- The first line should contain the edit distance, preceded by the text `"Edit distance = "`.
- Each subsequent line should contain a character from the first string or gap, followed by the paired character from the second string or gap, followed by the associated penalty. Use the character `'-'` to indicate a gap **in either string**. The string on the left should correspond to the first string in the file.

Here is a sample execution:

```
Edit distance = 7
A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1
```

Some sample files and the corresponding edit distances.

data file	distance
ecoli2500.txt	118
ecoli3000.txt	125

Demo: In Lab and submit to PolyLearn.

- Your program should be written to be as easy to understand as possible. There should be a separate table construction section and separate traceback section. Both clearly commented.
- There are test cases on PolyLearn. Following the same format you should develop your own test cases to ensure correctness.
- Your program must contain a "toggle" to allow you to print or not print the alignment and an easy way to change the input file that defines the problem. I do not care how this is done but it must be quick and easy to do during demo
- Be prepared to answer any questions about your program and demo it in lab on new test cases.

Hand in:

- Submit one or two actual java files that can be compiled and run. These are the files you will use to demo your program. Your code should be a simple as possible, use white space to clearly show the control structure and be **clearly commented**.