CSE317 Design & Analysis of Algorithms (Spring'19)

Assignment #2: Edit Distance

Max Marks: 20 *Due:* May 29

You are allowed to collaborate with other students provided that you do not exchange any written code. Write the solution on your own and mention the names of students you have collaborated with. Failure to do so may earn you a zero in this assignment.

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 $\{A, T, G, C\}$ of biological macromolecules (*Adenine (A), Thymine (T), Guanine (G), Cytosine (C)*) 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 uninformed 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.

```
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 make them have the same length). 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. Note that the scores are slightly different from the ones we used in the lectures.

opertion	**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:    A A C A G T T A C C
y:    T A A G G T C A - -
cost: 1+0+1+1+0+0+1+0+2+2 = 8
```

```
x:    A A C A G T T A C C
y:    T A - A G G T - C A
cost: 1+0+2+0+0+1+0+2+0+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 in fact optimal, so the edit-distance between the two strings is 7. Computing the edit-distance is a nontrivial computational problem because we must find the best alignment among exponentially many possibilities. For example, if both strings are 100 characters long, then there are more than 10^{75} possible alignments.
For this assignment, you need to develop a dynamic programming based solution similar to one develop in lecture but with the scoring presented in the table above.

API specification

Your program should consist of a class `EditDistance.java` with one static method to compute the optimal match between a pair of objects. The optimal solution is returned as a linked list of `Path` nodes. Each `Path` node stores the row and column of the element it represents, the total cost of the match from start to that point, and a pointer to the next path node. Note that the path starts at `row=a.length()-1` and `col=b.length()-1` and ends at the node with `row=0` and `col=0`.

```
public class EditDistance
------------------
  static Path match(String a, String b)  // return the optimal match between
                                         // the strings a and b
                                         // return null if either string is null
                                         // or if either string is length 0
```

```
  static void main(String[] args)    // read 2 strings from standard input.
                                     // compute and print the edit distance
                                     // between them and output an optimal
                                     // alignment and associated penalties.


public class Path
-----------------
  public int row, col;          // the row and column this node represents
  public int cost;              // the matching cost upto this point
  public Path next;             // the next node in the optimal path
```

Your program

Write a program `EditDistance.java` that conforms to the API above and whose main method reads, from standard input, two strings of characters, creates an `EditDistance` object for them, and computes the optimal matching between them using `EditDistance.match()`. (Although, in the application described, the characters represent genetic sequences, your program should handle any sequence of alphanumeric characters.) Your program should then print the edit distance between the two strings, and print out the optimal match 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, followed by the paired character from the second string, followed by the associated penalty. Use the character '-' to indicate a gap in either string.

Here is a sample execution:

```
% java EditDistance < example10.txt
                            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
```

Testing

The archive `sequence.zip` contains short test data files and actual genomic data files. To help you check the part of your program that finds the edit distance, the edit distances of `gene57.txt`, `stx1230.txt`, and `ftsa1272.txt` are 8, 521, and 758, respectively. To help you check the part of your program that generates the alignment, there are several short test files in the sequence directory whose alignments are easy to generate manually.

Note that the longer test sequences will require a great deal of memory. Be aware that you will not be able to run all the cases.

In addition, we require that you generate at least one input file of your own to be used for testing special cases. Create a new input file with some interesting property. Then test your code using your file and make sure your program behaves as expected.

Submit the file `EditDistance.java`. Also submit a zip file `extra-tests.zip` containing the test cases you created. Do not include the test cases you downloaded with the assignment. This must be a `zip` file, even if you submit only one test case, and your test cases should all be the root folder (not in a subfolder).