Com S 311 Spring 2022

Assignment 3: Computing the edit distance between sequences

Due: Thursday, March 24, 11:59 p.m.

Late Submission Due: Friday, March 25, 11:59 a.m. (25% penalty)

Separate Quiz Task Due: Monday, March 21, 11:59 p.m.

Introduction

This assignment gives you the opportunity to implement a dynamic programming algorithm for computing the edit distance between two sequences. This algorithm is useful for comparison of DNA sequences to find differences between them. The assignment will help you understand what it takes to develop a useful computer program. Your task in this project is to implement the algorithm described below. Please see the attached Java template code for the specification of each function to be implemented. You are required to use Java in this assignment.

1 Edit Distance and Sequence Alignment

We first give a formal definition of the edit distance between two sequences. Then we describe a dynamic programming algorithm for computing the edit distance between two sequences.

1.1 Formal definition

A similarity relationship between two sequences A and B can be represented by an alignment of the two sequences, an ordered list of pairs of letters of A and B. The alignment consists of substitutions, deletion pairs and insertion pairs. A substitution pairs a letter of A with a letter of B. A substitution is a match if the two letters are identical and a mismatch otherwise. A deletion pair involves a letter of A that corresponds to no letter of B, and an insertion pair involves a letter of B that corresponds to no letter of A. Deletion and insertion pairs are defined with regard to transformation of sequence A into sequence B. An alignment of A and B shows a way to transform A into B, where a letter of A is replaced by a letter of B in every mismatch, the letters of A in every deletion pair are deleted, and the letters of B in every insertion pair are inserted.

Below is an alignment of two DNA sequences AGCTACGTACACTACC and AGCTATCGTACTAGC. This alignment contains 13 matches, 1 mismatch, an insertion pair, and two deletion pairs. The top sequence is transformed into the bottom sequence by 4 operations: an insertion, two deletions, and a replacement.

```
AGCTA-CGTACACTACC
AGCTATCGTAC--TAGC
     ^     ^^  ^
     |     ||  |
     |     ||  a replacement
     |     ||
     |   two deletions
     |
an insertion
```

The cost of an alignment of A and B is the cost of the corresponding set of operations transforming A into B, which is defined as the sum of costs of each operation in the set, where the cost of a replacement of letter a by letter b is denoted by a substitution table $\sigma(a, b)$, and the cost of each insertion and deletion is denoted by a non-negative number r.

The edit distance between A and B is the minimum cost of operations that transform A into B. The edit distance is also the cost of an optimal alignment of A and B. Values for the parameters σ and r are specified by the user. A letter-independent substitution table is usually used for comparison of DNA sequences. For example, if the cost of each replacement is 20, and the cost of each insertion and deletion is $r = 15$, then the cost of the above set of operations is $65 = (1 * 15) + (2 * 15) + (1 * 20)$. Thus, the cost of the above alignment is also 65. The edit distance between the two sequences is 65, because the cost of the above operations is the lowest among all sets of operations that transform A into B, as verified by a computer program for computing the edit distance between two sequences. Thus, the above alignment is an optimal alignment.

1.2 A dynamic programming algorithm

Let $A = a_1a_2...a_m$ and $B = b_1b_2...b_n$ be two sequences (called strings in Java and C) of lengths m and n. The sequence A is stored on a computer with a **char** array AR of length $m + 1$ with a_i in $AR[i]$ for each $1 \leq i \leq m$ and with $AR[0]$ unused. The sequence B is stored with a **char** array of length $n + 1$. A technique called dynamic programming in computer science is used to compute the edit distance between A and B, and an optimal global alignment of A and B. For $0 \leq i \leq m$ and $0 \leq j \leq n$, let $A_i = a_{i+1}a_{i+2}...a_m$ denote a suffix of A and let $B_j = b_{j+1}b_{j+2}...b_n$ denote a suffix of B. Assume that A_m and B_n denote the empty sequence with no letter. In this technique, a matrix S is introduced: $S(i, j)$ is the minimum cost of all alignments of A_i and B_j. Note that $A_0 = a_1a_2...a_m = A$ and that $B_0 = b_1b_2...b_n = B$. Thus $S(0, 0)$ is the cost of an optimal alignment of A and B as well as the edit distance between A and B.

An efficient method for computing the matrix S is explained in the Appendix. Below we specify how the matrix is computed by using four formulas for the matrix. Each formula is used for the entries that meet the conditions given after the formula. Note that $\sigma(a_{i+1}, b_{j+1})$ is zero if a_{i+1} is identical to b_{j+1} and the replacement cost otherwise.

$S(m, n) = 0$,

$S(i, n) = S(i + 1, n) + r$ for $m > i \geq 0$,

$S(m, j) = S(m, j + 1) + r$ for $n > j \geq 0$,

$S(i, j) = \min\{S(i + 1, j + 1) + \sigma(a_{i+1}, b_{j+1}), S(i + 1, j) + r, S(i, j + 1) + r\}$

for $m > i \geq 0$ and $n > j \geq 0$.

The matrix can be computed in decreasing order of rows and then in decreasing order of columns. The complete computation is given in pseudocode in Figure 1. To help you understand the algorithm, two short sample sequences and their optimal alignment are provided along with all values of the matrix S below.

```
let a_1 a_2 ... a_m be a sequence of length m;
let b_1 b_2 ... b_n be a sequence of length n;
let r be a non-negative number;
let σ(a, b) be a non-negative number for a substitution pair (a, b);
S(m, n) = 0;
for j going from n − 1 downto 0
    {
      S(m, j) = S(m, j + 1) + r;
    }
for i going from m − 1 downto 0
    {
      S(i, n) = S(i + 1, n) + r;
      for j going from n − 1 downto 0
          {
            S(i, j) = min{S(i + 1, j + 1) + σ(a_{i+1}, b_{j+1}), S(i + 1, j) + r, S(i, j + 1) + r};
          } // inner loop
    } // outer loop
```

Figure 1. Computation of the matrix S in pseudocode.

Consider the following example.

Match score: 0 = sigma(a, b) if a == b

Mismatch penalty: 20 = sigma(a, b) if a != b

Gap penalty: 15 = r

Sequence A: ACGTCGAGCTA of length 11

Sequence B: ACCTCGACTA of length 10

Edit distance: 35

Number of differences: 2

Length of alignment: 11

Top row: ACGTCGAGCTA

Mid row: || ||||-|||

Bot row: ACCTCGA CTA

Computation of a few entries in the matrix S

S(11, 10) = 0,

S(11, 9) = S(11, 10) + 15 = 15,

S(11, 8) = S(11, 9) + 15 = 30,

.....

S(10, 10) = S(11, 10) + 15 = 15,

S(10, 9) = min{S(11, 10)+sigma(A, A), S(11, 9)+15, S(10, 10)+15}
= min{0+0, 15+15, 15+15} = 0,

S(10, 8) = min{S(11, 9)+sigma(A, T), S(11, 8)+15, S(10, 9)+15}
= min{15+20, 30+15, 0+15} = 15,

.....

S(0, 0) = min{S(1, 1)+sigma(A, A), S(1, 0)+15, S(0, 1)+15}
= min{35+0, 50+15, 50+15} = 35.

Complete Matrix S for sequences A (Seq1) and B (Seq2).
For 1 <= i <= 11, row i is marked with letter A[i], and
for 1 <= j <= 10, column j is marked with letter B[j].

		Seq2		A	C	C	T	C	G	A	C	T	A
		column	0	1	2	3	4	5	6	7	8	9	10
Seq1													
	row	0	35	50	45	60	75	90	105	120	135	150	165
A	row	1	50	35	30	45	60	75	90	105	120	135	150
C	row	2	65	50	35	30	45	60	75	90	105	120	135
G	row	3	60	45	30	15	30	45	60	75	90	105	120
T	row	4	75	60	45	30	15	30	45	60	75	90	105
C	row	5	90	75	60	45	30	15	30	45	60	75	90
G	row	6	75	80	65	50	35	30	15	30	45	60	75
A	row	7	90	75	60	45	30	15	20	15	30	45	60
G	row	8	105	90	75	60	45	30	15	0	15	30	45
C	row	9	120	105	90	75	60	45	30	15	0	15	30
T	row	10	135	120	105	90	75	60	45	30	15	0	15
A	row	11	150	135	120	105	90	75	60	45	30	15	0

An optimal alignment is found by a traceback procedure on the matrix S. An optimal alignment corresponds to a path from entry (m, n) to entry $(0, 0)$ in the matrix S. Let the current entry be a newly determined one. An optimal path is recovered by repeatedly determining an entry that is immediately after the current entry on the path. Thus the pairs of an optimal alignment are generated in a forward order with the first pair produced first. Initially, the current entry is $(0, 0)$.

Let the current entry be (i, j). The recurrences for S are used to determine the new entry and the next pair for the optimal alignment. If $i = m$ and $j = n$, then the traceback procedure terminates. Otherwise, if $j = n$ or $S(i, j) = S(i + 1, j) + r$, then the new entry is $(i + 1, j)$ and the next pair for the optimal alignment is a deletion pair $(a_{i+1}, -)$. Otherwise, if $i = m$ or $S(i, j) = S(i, j + 1) + r$, then the new entry is $(i, j + 1)$ and the next pair for the optimal alignment is an insertion pair $(-, b_{j+1})$. Otherwise, the new entry is $(i + 1, j + 1)$ and the next pair for the optimal alignment is a substitution pair (a_{i+1}, b_{j+1}).

```
let a_1 a_2 ... a_m be a sequence of length m;
let b_1 b_2 ... b_n be a sequence of length n;
let r be a non-negative number;
let σ(a, b) be a non-negative number for a substitution pair (a, b);
let S be the matrix computed previously;
let OA be empty;
i = 0;
j = 0;
while ( i ≤ m and j ≤ n )
   {
            if ( i == m and j == n )
               break;
            if ( j == n or i < m and S(i, j) == S(i + 1, j) + r )
               {
                  append pair (a_{i+1}, −) to OA;
                  i++;
                  continue;
               }
            if ( i == m or S(i, j) == S(i, j + 1) + r )
               {
                  append pair (−, b_{j+1}) to OA;
                  j++;
                  continue;
               }
            append pair (a_{i+1}, b_{j+1}) to OA;
            i++;
            j++;
   }
report the pairs of the alignment in OA;
```

Figure 2. Generation of an optimal alignment.

An optimal alignment with the cost 35 is produced by
the traceback method in Figure 2 on the example matrix S.

Because S(0, 0) = S(1, 1) + sigma(A, A), the first pair
in the alignment is (A, A).

Because S(1, 1) = S(2, 2) + sigma(C, C), the second pair
in the alignment is (C, C).
.....
Because S(6, 6) = S(7, 7) + sigma(A, A), the seventh pair
in the alignment is (A, A).

Because S(7, 7) = S(8, 7) + 15, the eighth pair
in the alignment is (G, -).

Because S(8, 7) = S(9, 8) + sigma(C, C), the ninth pair
in the alignment is (C, C).

Because S(9, 8) = S(10, 9) + sigma(T, T), the tenth pair
in the alignment is (T, T).

Because S(10, 9) = S(11, 10) + sigma(A, A), the eleventh pair
in the alignment is (A, A). This is the last pair.

The traceback terminates.
Complete alignment:
Top row: ACGTCGAGCTA
Mid row: || ||||-|||
Bot row: ACCTCGA CTA

The traceback procedure requires that the complete matrix be saved, which takes computer memory proportional to the product $m \times n$. Thus for two sequences of length 10,000, the algorithm takes computer memory in the order of 100,000,000 words. Because of the high computer memory requirement, only an optimal alignment of two sequences of at most a few thousand letters can be constructed on an ordinary computer using this algorithm. The time requirement of the algorithm is also proportional to the product $m \times n$. For two sequences of length 10,000, it takes less than a minute to compute the matrix S on an ordinary computer. Thus the space requirement of this algorithm is much more limiting than the time requirement.

Submission

This homework has an additional quiz task. To help you understand the algorithm in this assignment, a quiz task will be set up on Canvas. This task requires you to compute the cost matrix S on paper by applying the algorithm to the following sequences.

```
Sequence A: ACGAT of length 5
Sequence B: ATGT of length 4
```

You will earn the full credit for this quiz by submitting your answer to the quiz on Canvas and by studying the correct answer displayed on Canvas. Please note that this quiz task is in addition to the assignment submission task below.

Next, you are required to include, in your assignment submission, the source code for each of the classes: The template code segment is given in package cs311hw3.zip.

Write your class so that its package name is edu.iastate.cs311.hw3. Your source files (.java files) will be placed in the directory edu/iastate/cs311/hw3 (Mac or Linux) or edu\iastate\cs311\hw3 (Windows), as defined by the package specified above. Be sure to put down your name after the @author tag in each class source file. Your zip file should be named Firstname_Lastname_HW3.zip. You may submit a draft version of your code early to see if you have any submission problem on Canvas. We will grade only your latest submission.

You are allowed to discuss homework with classmates, but you must write your Java solution code alone, without consulting anyone. You are not allowed to post your Java solution source code in the common area on Piazza or in any public code repository, such as GitHub. However, you are allowed to share JUnit test code on Piazza. There is no partial credit for answering "I don't know" to this assignment.

Any concerns about grading should be expressed within one week of returning the assignment.

Appendix

In this section, we explain how the formulas for computing the matrix S are developed. If $i = m$ and $j = n$, then $S(m, n)$ is set to 0, denoting the cost of the empty alignment with no letters. For $i < m$ and $j = n$, B_n is the empty sequence with no letters. The alignment of A_i and B_n consists of $m - i$ deletion pairs $(a_{i+1}, -)(a_{i+2}, -)...(a_m, -)$, and its cost is $r \times (m - i)$. Thus, we have

$S(i, n) = r \times (m - i) = r \times (m - (i + 1)) + r = S(i + 1, n) + r.$

Similarly, for $i = m$ and $j < n$, we have

$S(m, j) = r \times (n - j) = r \times (n - (j + 1)) + r = S(m, j + 1) + r.$

For $i < m$ and $j < n$, let $P(A_i, B_j)$ denote an alignment of A_i and B_j with the minimum cost $S(i, j)$, that is, an optimal alignment of A_i and B_j. The first aligned pair of $P(A_i, B_j)$ has to be one of the following aligned pairs: a substitution pair (a_{i+1}, b_{j+1}), a deletion pair $(a_{i+1}, -)$, and an insertion pair $(-, b_{j+1})$. If the first aligned pair of $P(A_i, B_j)$ is the substitution pair (a_{i+1}, b_{j+1}), then the portion of $P(A_i, B_j)$ after the first substitution pair (a_{i+1}, b_{j+1}) is an alignment of A_{i+1} and B_{j+1} with the minimum cost $S(i+1, j+1)$, because $P(A_i, B_j)$ is an alignment of A_i and B_j with the minimum cost. In this case, alignment $P(A_i, B_j)$ consists of the substitution pair (a_{i+1}, b_{j+1}) and alignment $P(A_{i+1}, B_{j+1})$. Thus the cost of $P(A_i, B_j)$ is equal to the cost of $P(A_{i+1}, B_{j+1})$ plus $\sigma(a_{i+1}, b_{j+1})$, that is,

$S(i, j) = S(i + 1, j + 1) + \sigma(a_{i+1}, b_{j+1}).$

If the first aligned pair of $P(A_i, B_j)$ is a deletion pair $(a_{i+1}, -)$, then the portion of

$P(A_i, B_j)$ after the first deletion pair $(a_{i+1}, -)$ is an alignment of A_{i+1} and B_j with the minimum cost $S(i+1, j)$, because $P(A_i, B_j)$ is an alignment of A_i and B_j with the minimum cost. In this case, alignment $P(A_i, B_j)$ consists of the deletion pair $(a_{i+1}, -)$ and alignment $P(A_{i+1}, B_j)$. Thus the cost of $P(A_i, B_j)$ is equal to the cost of $P(A_{i+1}, B_j)$ plus the cost r of the deletion pair $(a_{i+1}, -)$, that is,

$$S(i, j) = S(i+1, j) + r.$$

Similarly, if the first aligned pair of $P(A_i, B_j)$ is an insertion pair $(-, b_{j+1})$, we have

$$S(i, j) = S(i, j+1) + r.$$

By the definition of the matrix S, $S(i, j)$ is the minimum cost of all alignments of A_i and B_j. The expression on the right hand side of each inequality below is the cost of an alignment of A_i and B_j. Thus, each inequality holds.

$$S(i, j) \leq S(i+1, j+1) + \sigma(a_{i+1}, b_{j+1}),$$

$$S(i, j) \leq S(i+1, j) + r,$$

$$S(i, j) \leq S(i, j+1) + r.$$

Thus we conclude that for $i < m$ and $j < n$,

$$S(i, j) = \min\{S(i+1, j+1) + \sigma(a_{i+1}, b_{j+1}), S(i+1, j) + r, S(i, j+1) + r\}.$$