

Name: Michael Rogers

ID: 105667404

**CSCI 3104**  
**Problem Set 7**

**Profs. Grochow & Layer**  
**Spring 2019, CU-Boulder**

- Multiple string alignment. As in class, we consider the operations of substitution (including the zero-cost substitute-a-letter-for-itself “no-op”), insertion, and deletion. An alignment of three strings  $x, y, z$  (of lengths  $n_x, n_y, n_z$ , respectively) is an array of the form:

$$\begin{array}{cccccccccc}
 x_1 & x_2 & - & x_3 & - & - & x_4 & \dots & x_{n_x} & - \\
 - & y_1 & y_2 & - & - & y_3 & y_4 & \dots & - & y_{n_y} \\
 z_1 & - & - & z_2 & z_3 & - & - & \dots & z_{n_z} & -
 \end{array}$$

That is, in the first row  $x$  appears in order, possibly with some gaps, in the second row  $y$  does, and in the third row  $z$  does. We define the cost of such an alignment as follows. The cost of a column

$$\begin{array}{c}
 x_2 \\
 y_1 \\
 -
 \end{array}$$

is the sum-of-pairs cost, e.g., in the preceding example we look at the cost of aligning  $x_2$  with  $y_1$  (a substitution, costing either 0 or 1 depending on whether  $x_2 = y_1$  or not), the cost of aligning  $x_2$  with  $-$  (a deletion, costing 1), and the cost of aligning  $y_1$  with  $-$  (another deletion), for a total cost of  $c_{subs} + 2c_{del} = 3$  for this one column of the alignment. The total cost of the alignment is the sum of the costs of its columns. Given three strings  $x, y, z$ , the goal of Multiple String Alignment is to find a minimum-cost alignment.

I don't know

Name: Michael Rogers

ID: 105667404

**CSCI 3104**  
**Problem Set 7**

**Profs. Grochow & Leyer**  
**Spring 2019, CU-Boulder**

---

- (a) (15 pts) Give an efficient (polynomial-time) algorithm for finding optimal alignments of three strings. Describe your algorithm in pseudocode and English, and prove an upper bound on its running time; you do not need to prove correctness (but you will only receive full credit if your algorithm is correct!). Hint 1: start with a recursive algorithm, where the recursion has 7 cases depending on what the last column of the alignment looks like:

$$\begin{pmatrix} x_{n_x} \\ y_{n_y} \\ z_{n_z} \end{pmatrix}, \begin{pmatrix} - \\ y_{n_y} \\ z_{n_z} \end{pmatrix}, \begin{pmatrix} x_{n_x} \\ - \\ z_{n_z} \end{pmatrix}, \begin{pmatrix} x_{n_x} \\ y_{n_y} \\ - \end{pmatrix}, \begin{pmatrix} - \\ - \\ z_{n_z} \end{pmatrix}, \begin{pmatrix} - \\ y_{n_y} \\ - \end{pmatrix}, \begin{pmatrix} x_{n_x} \\ - \\ - \end{pmatrix}$$

Hint 2: Your recursive algorithm will likely not be very efficient; what technique can you use from class to improve its efficiency?

I don't know

Name: Michael Rogers

ID: 105667404

**CSCI 3104**  
**Problem Set 7**

**Profs. Grochow & Layer**  
**Spring 2019, CU-Boulder**

---

- (b) (7 pts) Consider generalizing your approach to part (a) to  $k$  strings instead of 3. How many cases would there be to consider in the recursion? What runtime would the algorithm have? You do not need to give the algorithm, but must argue persuasively that your answers are correct, given your answer to part (a).

I don't know

CSCI 3104  
Problem Set 7

Profs. Grochow & Leyer  
Spring 2019, CU-Boulder

2. The most efficient version of the preceding approach we know of for three strings takes an amount of time which becomes impractical as soon as the strings have somewhere between  $10^4$  and  $10^5$  characters (which are still actually fairly small sizes if one is considering aligning, e. .g, genomes). Because of this, people seek faster heuristic algorithms, and this is even more true for aligning more than three strings. One natural approach to faster multi-string alignment is to first consider optimal pairwise alignments and somehow use this information to help find the multi-way alignment. Here we show that naive versions of such a strategy are doomed to fail.

Given an alignment of three strings, we may consider the 2-string alignments it induces: just consider two of the rows of the alignment at a time. If both of those rows have a gap in some column, we treat the pairwise alignment as though that column doesn't exist. For example, the alignment of  $x$  and  $y$  induced by the figure at the top of Q1 would be

$$\begin{array}{ccccccccccc} x_1 & x_2 & - & x_3 & - & x_4 & \dots & x_{n_x} & - & & \\ - & y_1 & y_2 & - & y_3 & y_4 & \dots & - & y_{n_y} & & \end{array}$$

Note that the column between  $x_3$  and  $y_3$  got deleted because it consisted of two gaps.

Name: Michael Rogers

ID: 105667404

**CSCI 3104**  
**Problem Set 7**

**Profs. Grochow & Laver**  
**Spring 2019, CU-Boulder**

---

- (a) (5pts) Give an example of three strings  $x, y, z$  such that the optimal alignment of  $x$  with  $y$  does not appear in *any* optimal alignment of  $x, y, z$ , and the same is true for optimal alignments of  $x, y$  and of  $y, z$ . Prove your example has this property. (An example where any optimal alignment of  $x, y, z$  does not contain an optimal alignment of  $x, y$ —but may contain optimal alignments of  $x, z$  and/or  $y, z$ —will earn you partial credit.)

I don't know

Name: Michael Rogers

ID: 105667404

**CSCI 3104**  
**Problem Set 7**

**Profs. Grochow & Laver**  
**Spring 2019, CU-Boulder**

---

- (b) (7 pts) Prove that the gap between the cost of the pairwise optimal alignments and the pairwise alignments induced by an optimal 3-way alignment can be arbitrarily large. More symbolically, let  $d_{xy}$  denote the cost of an optimal alignment of  $x$  and  $y$ , and for an alignment  $\alpha$  of  $x, y, z$ , let  $d_{xy}(\alpha)$  denote the cost of the pairwise alignment of  $x$  and  $y$  which is induced by  $\alpha$ . Your goal here is: for any  $c > 0$ , construct three strings  $x, y, z$  such that  $d_{xy}(\alpha) > d_{xy} + c$  for any three-way alignment  $\alpha$ . (Suppose all three strings have length  $n$ ; how big can you make the gap  $c$  as a function of  $n$ , asymptotically?)

I don't know

Name: Michael Rogers

ID: 105667404

**CSCI 3104**  
**Problem Set 7**

**Profs. Grochow & Layer**  
**Spring 2019, CU-Boulder**

---

3. (14 pts) Give an efficient algorithm to compute the *number* of optimal solutions to the Knapsack problem. Recall the Knapsack problem has as its input a list  $L = [(v_1, w_1), \dots, (v_n, w_n)]$  and a threshold weight  $W$ , and the goal is to select a subset  $S$  of  $L$  maximizing  $\sum_{i \in S} v_i$ , subject to the constraint that  $\sum_{i \in S} w_i \leq W$ . For this problem, you will count the number of such optimal solutions. Your algorithm should run in  $O(nW)$  time. If you only give an inefficient recursive algorithm, you can still receive up to 6 pts. (If you are having trouble solving this problem, you may want to start by developing an inefficient recursive algorithm and then seeing how to improve it using techniques from class.)

I don't know

Name: Michael Rogers

ID: 105667404

**CSCI 3104**  
**Problem Set 7**

**Profs. Grochow & Laver**  
**Spring 2019, CU-Boulder**

---

4. Consider the following variant of string alignment: given two strings  $x, y$ , and a positive integer  $L$ , find all contiguous substrings of length at least  $L$  that are aligned (using no-ops = substituting a letter for itself) in some optimal alignment of  $x$  and  $y$ . Assume the costs of substitution, insertion, and deletion are given by constants  $c_{subs}, c_{ins}, c_{del}$ , and that the cost of substituting a letter for itself is zero.

(a) (2 pts) Show that the output here contains at most  $O((n - L)^3)$  substrings.

I don't know



Name: Michael Rogers

ID: 105667404

**CSCI 3104**  
**Problem Set 7**

**Profs. Grochow & Layer**  
**Spring 2019, CU-Boulder**

---

- (b) (10 pts) Give an algorithm that solves this problem. You may use/modify/refer to the pseudocode for Knapsack from the lecture notes.

I don't know