Problem Set 5

All parts are due on December 8, 2016 at 11:59PM. Please download the .zip archive for this problem set. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Note that due to MIT policy on end of semester assignments, **you may use at most 1 slack day on this problem set**, even if you have more than one remaining.

Part A

Problem 5-1. [35 points] **Fantastic Beasts**

You are aiding Newt Scamander, the Wizarding World's pre-eminent magizoologist, in his quest to discover and understand magical creatures. You have managed to sequence a gene, which is a string of a 4 letter alphabet $\{A, T, G, C\}$, from a Hippogriff, and you want to answer the age-old question of whether a Hippogriff is more eagle or more horse by comparing this gene to the existing and well-understood horse and eagle genes.

You will measure how "close" two gene sequences are by computing the optimal "score" of an "alignment" of the two genes. An ***alignment*** of two strings a and b is a pair of alignment strings a^+ and b^+ which contain gap characters '_' such that

1. $|a^+| = |b^+|$;
2. for each i with $0 \le i \le |a^+| - 1$, at most one of $a^+[i]$ and $b^+[i]$ equals '_';
3. removing all '_' symbols from a^+ yields a; and
4. removing all '_' symbols from b^+ yields b.

For example, if we have two genes $a = ATAGCATGC$ and $b = ACTAGCTGC$, then one possible alignment of these genes is as follows:

$$a^+ = ATAG\_CAT\_GC,$$
$$b^+ = A\_CTAG\_CTGC.$$

The ***score*** of a particular alignment $\{a^+, b^+\}$ of two genes is the sum of the "character-match scores" for each index of the alignment:

$$\texttt{score}(a^+, b^+) = \sum_{i=0}^{|a^+|-1} w(a^+[i], b^+[i]),$$

where the ***character-match scores*** $w(i,j)$ are given by the following table:

	A	T	G	C	'_'
A	0	1	2	3	4
T	1	0	3	2	4
G	2	3	0	1	4
C	3	2	1	0	4
'_'	4	4	4	4	n/a

Different alignments of the same two genes can produce different scores. We define an ***optimal alignment*** of two genes to be an alignment of minimum possible score. For example, the example alignment above has a score of 25, while the optimal alignment of the two genes $ATAGCATGC$ and $ACTAGCTGC$ actually has score 8:

$$a^+ = A\_TAGCATGC,$$
$$b^+ = ACTAGC\_TGC.$$

(a) [2 points] Design a dynamic programming algorithm that, given two genes a and b, each of length $\Theta(n)$, finds the score of their optimal alignment in $O(n^2)$ time and $O(n^2)$ space.

(b) [3 points] Describe how to modify your algorithm from (a) to use only $O(n)$ space.

You have now decided that you would like to know not only which gene is more closely related, but also the actual optimal alignment of base pairs, and not just the scores.

(c) [5 points] Describe how to modify your algorithm from (a) to compute the optimal alignment in $O(n^2)$ time and $O(n^2)$ space.

(d) [15 points] Describe how to modify your algorithm from (b) to compute the optimal alignment in $O(n^2)$ time and $O(n)$ space.

Hint: To solve this problem, you will simultaneously need to work forwards (aligning prefixes of the strings) and backwards (aligning suffixes of the strings), until they meet on the main diagonal of the DP matrix. Then divide and conquer in two of the submatrices, to obtain the recurrence $T(p) = 2T(p/4) + O(p)$ where p is the total number of entries in the matrix (i.e., the product of the lengths of a and b).

(e) [10 points] Unfortunately, you find that your solution to part (c) is taking too long to run on the DNA sequences (whose lengths are in the millions). You decide to modify your program to approximate the optimal alignment by restricting the number of gaps allowed in each alignment string a^+ and b^+ to 100. Describe how to modify your algorithm from part (c) to satisfy this additional constraint. What is the running time of the resulting algorithm?

Problem 5-2. [20 points] **Rye Elections**

Prof. Caufield has decided to run in the 2020 U.S. presidential race, and has appointed you as his campaign manager. He wants to campaign in the state of Massachusetts, which (thanks to rising oceans) has become a line of n cities, $c_0, c_1, \ldots, c_{n-1}$. Prof. Caufield's campaign consists of running television advertisements in specific cities. Each city c_i charges a price p_i to broadcast an advertisement. Furthermore, the residents of Massachusetts love nothing more than to talk amongst each other and gossip, and so if Holden runs a television ad in city c_i, its k neighboring cities on each side $(c_{i-k}, \ldots, c_{i-1}, c_i, c_{i+1}, \ldots, c_{i+k})$ all hear about Prof. Caufield's exciting policies.

Can you design an algorithm to choose which cities to advertise in, to reach all of the cities in Massachusetts, for the minimum total price?

(a) [5 points] Let's first consider the case where $k = 1$, and thus running an ad in a city reaches itself and both of its neighboring cities. Prof. Caufield comes up with the following algorithm: until all cities have been reached, select the cheapest unreached city and run an advertisement there. Does this algorithm minimize the cost of reaching all of the n cities? Prove correctness or provide a counterexample.

(b) [10 points] For the $k = 1$ case, design an algorithm that computes the subset of cities in which Prof. Caufield should run advertisements that minimizes the total cost and still reaches every city, using $O(n)$ time.

(c) [5 points] Describe how to modify your algorithm for the case of general k, while running in $O(nk)$ time.

Part B

Problem 5-3. [45 points] **Halloween**

Prof. Neko is really excited about Google's Halloween Doodle this year.[1] She now wants to beat the world record. In particular, Prof. Neko needs to solve the following problem.

There are g ghosts on the screen, labeled $1, 2, \ldots, g$. Each ghost has a sequence of game moves needed to eliminate it. There is a small constant number m of possible game moves, which we denote $A, B, C, \ldots$ below. (In the game, they are —, |, $\vee$, $\wedge$, and ↯.) We denote the required move sequence of ghost i by $s_i = (s_i[0], s_i[1], \ldots, s_i[\ell_i - 1])$ where ℓ_i is the length of the sequence s_i. Note that different ghosts can have sequences of different length. Prof. Neko now needs to find a sequence of moves so that all of the g ghosts disappear. A ghost ***disappears*** when all of the moves in its sequence have been played, in that order, potentially with other moves in between.

It is not hard to come up with a sequence that makes all ghosts disappear. We can simply play the moves of the first ghost, then the moves of the second ghost, etc. Note that this can take up to

[1]Try it out yourself! `https://www.google.com/doodles/halloween-2016`

$\sum_{i=1}^{g} \ell_i$ moves in total, but there might be much shorter move sequences. In particular, consider the following example:

Ghost 1: (A, B, B, B)
Ghost 2: (C, B, B, B, B)

First playing all the moves of Ghost 1 followed by the moves of Ghost 2 takes nine moves in total. Instead, we could play (A, C, B, B, B, B), which is only six moves and still makes all ghosts disappear. After making this realization, Prof. Neko now wonders how to find the *shortest* possible move sequence for a given set of ghosts.

(a) [20 points] First, we consider the $g = 2$ case. Implement a dynamic programming algorithm that computes the shortest possible move sequence that makes both ghosts disappear in the `double_kill` method. The algorithm should run in $O(\ell_1 \cdot \ell_2)$ time. *Hint:* Use parent pointers to reconstruct the optimal alignment.

(b) [10 points] Next, extend your algorithm to the $g = 3$ case in the `triple_kill` method. This algorithm should run in $O(\ell_1 \cdot \ell_2 \cdot \ell_3)$ time.

(c) [5 points] Another way to view this problem is as a graph. Describe how you would formulate the problem from part (b) as a shortest-paths problem in a graph. Your algorithm should still run in $O(\ell_1 \cdot \ell_2 \cdot \ell_3)$ time.

(d) [5 points] For either the graph or the dynamic programming approach, extend your algorithm so that it applies to an arbitrary, fixed number of ghosts g, and runs in $O(\prod_{i=1}^{g} \ell_i)$ time.

(e) [5 points] In the game, there is a special symbol $\lightning$ that has a different behavior from the other symbols: it can only be played when one of the ghosts currently has it at the next symbol in its sequence, and when played, it eliminates the next symbol of *all* g ghosts. Describe how to modify your dynamic program from part (d) to handle this behavior of $\lightning$.