COMP 462/561 – Computational Biology Methods
Midterm examination – October 16th 2019, 4:05 – 5:25

NAME: ____ MCGILL ID #: __________________________

Instructions:

- The exam has 4 questions
- Answer directly on the exam
- This is an open book exam.
- No calculators, computers, or any electronics allowed.

1) Short answer questions (44 points)

a) (6 points) In bacteria, the average length of proteins in the is 320 amino acids. What is the average length of a the protein-coding portion of a gene (in nucleotides)?

3 * 320 = 960

b) (8 points) In the Blast algorithm, an ungapped extension phase is performed before doing the gapped extension phase. Why not skip the ungapped extension phase and go directly to the gapped extension phase?

This would make the algorithm very slow, as the ungapped extension is a process that allows the algorithm to decide not to perform the csotly gapped extension phase when the result of the ungapped extension is not good enough. However, skipping the ungapped extension would not have any negative impact on the accuracy of the results.

c) (8 points) List at least <u>three</u> possible consequences of a nucleotide <u>substitution</u> within the coding portion of a gene on the amino acid sequence of the protein it encodes?

- No change to the protein sequence (Synonymous mutation)
- Change of a single amino acid (Non-synonymous mutation)
- Creation of a premature STOP codon (nonsense mutation)
- Alteration of START codon, results in a shorter protein and (possibly) a frameshift
- (in eukaryotes) The mutation may alter a splice site, making the improperly spliced.

d) (12 points) Consider an evolutionary process that follows exactly the molecular clock assumption, i.e. mutations occur at a constant rate, and suppose that we are given 6 very long sequences so that we can estimate with perfect accuracy the number of years of divergence between any two given species. The resulting distance matrix is:

	A	**B**	**C**	**D**	**E**	**F**
A	-	10	8	10	2	6
B	10	-	10	4	10	10
C	8	10	-	10	8	8
D	10	4	10	-	10	10
E	2	10	8	10	-	6
F	6	10	8	10	6	-

Draw the rooted phylogenetic tree and branch lengths that best match this distance matrix.
Hint: Although you could apply the UPGMA algorithm and get the correct answer, you can probably guess the correct tree.

```
            -1-  A
        --2--+
        |   -1-  E
     -1-+
    |   ---3---  F
--1-+
|   ------4----  C
|
|           -2-  B
-----3------+
            -2-  D
```

f) (12 points) Consider the problem discussed in class to introduce Hidden Markov Models, where a person walks randomly in the city of Montreal, across different neighrborhoods (S = {F, E, C}), hearing different greetings (alphabet Sigma = {b, h, n, a}) every minute. Assume that the Emission and Transition probability matrices are the following:

Emission =

	b	**h**	**n**	**a**
F	1	0	0	0
E	0	1	0	0
C	0.3	0.2	0.25	0.25

Transition:

	F	**E**	**C**
F	0.9	0	0.1
E	0	0.9	0.1
C	0.5	0.5	0

Initial State Probability: 0.333 for all states

Consider the following sequence of observations:

```
             X =  b   b   b   b   b   h   h   h   h   h   h

Optimal path P =  F   F   F   F   C   E   E   E   E   E   E
```

What is the path *P* for which Pr[*P*,*X*] is maximized? Write your answer above, and explain your answer below.

Hint: Although you could use the Viterbi algorithm to answer the question, you should instead take a careful look at the emission and transition probability tables, which should suggest an easier approach.

Explanation (max 5 lines):

Observing b is much more likely from state F than from E or C. Observing h is much more likely from E than from F or C. So fom the perspective of emission probabilities, we'd like to be in state F for the first 5 steps, and then transition to F for the last 6 steps. However the trasition probability from F to E is zero. So we have to go through the C state. The only question is then: when should the transition F->C->E happen? It should happen as shown above, because Pr[b|C] > Pr[h|C].

2) Short-ish answer questions (14 points)

The global pairwise alignment problem with linear gap penalty is meaningful and interesting when the score *c* of gap is neither too large nor too small in comparison to the score of a match and mismatches. In this question, we will explore what happens to the solutions to the optimal alignment problem when we go beyond those values. We will work under the following assumptions:

- We are using a substitution cost matrix *M* where $M(a,a) = +1$ and $M(a,b) = -1$ for $a \neq b$.
- We are using a linear gap penalty: score(*L*) = $c*L$, where *L* is the length of the gap and *c* is the gap cost.
- IMPORTANT: The two sequences being aligned have *exactly* the same length *n*.

a) (7 points) When *c* becomes too large negatively, the solution to the alignment problem beomes trivial, because the optimal alignment is always an alignment with no gaps at all, irrespective of the content of the two sequences. Determine at what values of *c* this guaranteed to happen, and explain your reasoning. The value of *c* may depend on *n*.

We first note that if the two sequences have exactly the same length, then the number of gaps inserted in each sequence must be equal, i.e. the total number of gaps in the alignment must be either 0, 2, 4… The smallest number of gaps in a non-gap-free alignment is thus 2.

The highest score for a gap-containing alignment is achieved on an example like this one:

```
S = ACGTACGT…ACGTACGT
T = CGTACGT…ACGTACGTA
```

If c is not too large, the optimal alignment is

```
S = ACGTACGT…ACGTACGT-
T = -CGTACGT…ACGTACGTA
```

which has score (n-1) + 2 c

When c becomes too large, we get an alignment with no gaps at all, with score –n.
So we solve for c in the inequality $-n > (n-1) + 2c$:
➔ $c < -n + 1/2$

b) (7 points) When *c* becomes too small (i.e. close to zero), the solution to the alignment problem also becomes less interesting (but not trivial), because the optimal alignment never contains any mismatches at all, irrespective of the content of the two sequences. Determine at what value of *c* this happens, and explain your reasoning.

This happens when the score of a mismatch (-1) is worse than the score of 2 gaps (2c), because when that's the case we're always better off replacing any mismatches with two gaps, e.g.

```
A  ➔  A-
G     -G
```

So the answer is $2c > -1$, i.e. $c > -1/2$

3) Prefix-suffix alignment (20 points)

Consider the following variant of the pairwise sequence alignment problem.
Given: - Sequences $S = s_1\, s_2\, \ldots\, s_m$, and $T = t_1\, t_2\, \ldots\, t_n$,
- Scoring scheme using an linear gap penalty c and a substitution matrix M

Find: The prefix $s_1 \ldots s_i$ of S and the suffix $t_j \ldots t_n$, of T, that, when optimally aligned, yield the highest possible alignment score.

For example, for

```
S = A T A C A G A T A A G C
T = G A A G T A T G A T G A T
```

with $c = -1$ and $M(a,b) = +1$ if $a=b$ and -1 otherwise, the optimal alignment has score +3, which is obtained with $i = 8$ and $j = 6$, as shown in bold below:

```
          A T - A C A G A T A A G C
G A A G T A T G A T - G A T
```

Task: Describe an algorithm to solve the prefix-suffix exactly. Your algorithm must run in time O(m n). The output should be (1) the score of the optimal alignment, and (2) the values of i (position where the prefix of S ends) and j (position where the suffix of T starts). You don't need to output the optimal alignment itself.

We achieve this with a modified version of the Needleman-Wunsch algorithm, where the gaps inserted in S before the first nucleotide are free, and the gaps inserted in T after the last nucleoide are also free. This requires a modification to the initialization of the dyn. Prog. Algo, and another for starting the traceback.

Let X be the dyn. Prog. Table.

X(0,j) = 0 🡰 First modification
X(i,0) = c*i

X(i,j) = same as in NW algo

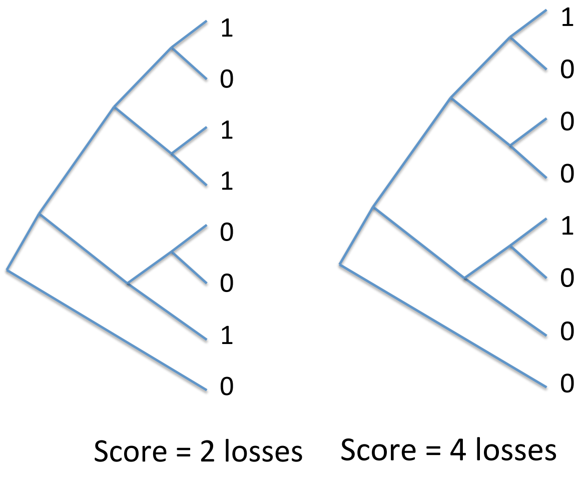
Initiate traceback from argmax{X(i,|T|): i =1…|S|} 🡰 Second modification
End trace back when we reach i=0.

4) Maximum Parsimony algorithms (20 points)

Consider the following problem, which is a variant of the Maximum Parsimony problem.

We have a set of *n* species placed at the leaves of a known phylogenetic tree *T*. Each species *s* is known to have or not have a particular binary propery P (for example, the ability to perform photosynthesis). This is denoted by *P(s)* = 1 or *P(s)* = 0. We assume that the property *must have arisen only once* during evolution (either before the root of *T*, or at some point along a single branch of *T*). However, the property may have been *lost multiple times* independently along different branches of the tree.

Task: Describe a polynomial time algorithm that computes the minimum number of losses (edges where P(parent)=1 and P(child)=0) needed to explain the data at the leaves of the tree, subject to the constraint that *there can be at most one gain (edges where P(parent)=0 and P(child)=1)*. See examples to the righ.



There are many ways to solve this question. Here's one.

First, we find the most recent common ancestor of all the leaves where P=1. In the two examples above, this is the left child of the root. This node is where the property first appeared. [It cannot have appeared later than this, because it would then require two gains to explain the data at the leave. If it appeared earlier than this, it would be sub-optimal as it would require some additional losses. Call this node X. All internal nodes located outside of the subtree rooted at X will be assigned value 0. To label the internal nodes in the subtree rooted at X, we used a modified version of the Sankoff algorithm, in which gains are disallowed:

Su[1] = max(Sv[0]+1, Sv[1]) + max(Sw[0]+1, Sw[1])
Su[0] = Sv[0] + Sw[0]

The rest of the algorithm is the same as before.

Note: An alternate approach to label the nodes of the subtree rooted at X is to use the following rule. To label node u, choose 0 is all the leaves of the subtree rooted at u have value 0. If that's not the case, set u to 1.

This page is left blank intentionally. Use it if you need extra space.

This page is left blank intentionally. Use it if you need extra space.