CS466: Introduction to Bioinformatics **Fall 2018**

Problem Set 1

Handed out: September 10, 2018 *Due: September 17, 2018*

Instructions: This homework assignment consists of four questions worth a total of 100 points. In addition, there is a bonus question on the third page worth an additional 20 points. These questions are based on the material covered in lectures 1 to 4.

1. **Asymptotic Running Time** [10 points]

   Consider the following running time functions, where $n > 0$.

n^2	n^3	$\sqrt{n}$	$n^2 \log(n)$	$n \log(n)$	$n!$	2^n	n
$n(n+1) - n^2$	$n + n^2$	$n \log(n^2)$	$n^3 - n^2$	1	$n^2 - n$	n^n	10,000,000

   a. Identify groups of functions such that for any pair $(f(n), g(n))$ of functions in the same group it holds that both $f(n) = O(g(n))$ and $g(n) = O(f(n))$. Note that some groups will contain a single function. [5 points]

   Hint: For example, $f(n) = 3n$ and $g(n) = n$ would be in the same group, as $f(n) = 3n = O(n) = O(g(n))$ and $g(n) = n = O(3n) = O(f(n))$.

   Answer:

Group 1:	1; 10,000,000	$O(1)$
Group 2:	$\sqrt{n}$	$O(\sqrt{n})$
Group 3:	n; $n(n+1) - n^2$	$O(n)$
Group 4:	$n \log(n)$; $n \log(n^2)$	$O(n \log n)$
Group 5:	n^2; $n + n^2$; $n^2 - n$	$O(n^2)$
Group 6:	$n^2 \log(n)$	$O(n^2 \log n)$
Group 7:	n^3; $n^3 - n^2$	$O(n^3)$
Group 8:	2^n	$O(2^n)$
Group 9:	$n!$	$O(n!)$
Group 10:	n^n	$O(n^n)$

   b. Arrange the resulting Big Oh running time groups in order from fastest to slowest. [5 points]

   Answer: Group 1 $\subset \ldots \subset$ Group 10.

2. **Sequence Alignment** [45 points]

   Consider two DNA sequences $\mathbf{v}$ = TAGATA and $\mathbf{w}$ = GTAGGCTTAAGGTTA. In this exercise, we will align the two sequences using a score of +1 for a match, -1 for a mismatch, and -1 for a insertion/deletion (i.e. a gap penalty of 1). We will use three different alignment algorithms. In each case, show the alignment (in the form of a matrix), the optimal score and the dynamic programming table. If there are multiple optimal alignments, you can report any one of them.

Answer: You can check the DP matrix through online demos such as: https://valiec.github.io/AlignmentVisualizer/index.html

a. Use global alignment. That is, align all of **v** with all of **w**. [15 points]

Answer:

v	-	T	A	G	-	-	-	-	A	-	-	-	T	-	A
w	G	T	A	G	G	C	T	T	A	A	G	G	T	T	A

Score: 6-9 = -3

b. Use fitting alignment. That is, find an alignment of **v** and a substring of **w** with maximum global alignment score. [15 points]

Answer:
Solution 1:

v	-	T	A	G	A	-	-	T	A	-	-	-	-	-	-
w	G	T	A	G	G	C	T	T	A	A	G	G	T	T	A

Score: 5-3=2

Solution 2:

v	-	-	-	-	-	-	-	T	A	-	G	A	T	-	A
w	G	T	A	G	G	C	T	T	A	A	G	G	T	T	A

Score: 5-3=2

c. Use local alignment. That is, find an alignment of a substring of **v** and a substring of **w** with maximum global alignment score. [15 points]

Answer:

v	-	T	A	G	-	-	-	-	-	-	-	-	-	-	-
w	G	T	A	G	G	C	T	T	A	A	G	G	T	T	A

Score: 3

3. **Longest Common Subsequence (LCS)** [35 points]

The longest common subsequence (LCS) problem is the problem of finding the longest subsequence common to two given sequences $\mathbf{v} = v_1, \ldots, v_m$ and $\mathbf{w} = w_1, \ldots, w_n$. The characters of a *subsequence* $\mathbf{v}'$ appear in the same relative order as in their originating sequence **v**, but are not necessarily contiguous in **v**. For example, the LCS for input sequences **v** = ABCDGH and **w** = AEDFHR is ADH. The LCS for input sequences **v** = AGGTAB and **w** = GXTXAYB is GTAB. Based on your understanding of the problem statement answer the following questions.

a. What is the LCS between sequence $\mathbf{v}$ = ABGXINDGOEPQ and $\mathbf{w}$ = ZZXBFICNAGGO? [10 points]

Answer: XINGO or BINGO

b. The LCS problem has optimal substructure. Understand why and write down a recurrence to solve it. Similarly to the lectures, use the notation $\mathbf{v}_i$ to denote the sequence $v_1, \ldots, v_i$ and $\mathbf{v}_0$ to denote the empty sequence. In the same vein, $\mathbf{w}_j$ denotes the sequence $w_1, \ldots, w_j$ and $\mathbf{w}_0$ is the empty sequence. [15 points]

Hint: Look carefully at the recurrence for edit distance, and think about which case should not be allowed. Do not forget that we want the *longest* common subsequence rather than the *smallest* edit distance. Also, do not forget to specify the base case(s) in your recurrence!

Answer: Let $s[i, j]$ denote the length of the longest common subsequence of $\mathbf{v}_i$ and $\mathbf{w}_j$, where $i \in \{0, \ldots, n\}$ and $j \in \{0, \ldots, m\}$. Thus, the length of the longest common subsequence of $\mathbf{v}$ and $\mathbf{w}$ is given by $s[m, n]$. Clearly, $s[0, 0] = 0$. We have the following recurrence.

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1, j], & \text{if } i > 0, \\ s[i, j-1], & \text{if } j > 0, \\ s[i-1, j-1] + 1, & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases} \tag{1}$$

c. Give pseudocode for computing the length of a LCS by solving the recurrence using dynamic programming (DP). Describe how an actual LCS can be computed from the DP table. [5 points]

Hint: See lecture slides for examples of pseudocode.

Answer:

```
function lengthLCS(v = v_1, ..., v_m, w = w_1, ..., w_n)
    s[0,0] ← 0
    for i ← 1 to m
        s[i,0] ← 0
    for j ← 1 to n
        s[0,j] ← 0
    for i ← 1 to m
        for j ← 1 to n
            if v_i = w_j
                s[i,j] ← s[i-1,j-1] + 1
            else
                s[i,j] ← max(s[i,j-1], s[i-1,j])
    return s[m,n]
```

An actual LCS $\mathbf{x}$ can be computed from s by backtracking from cell $s[m, n]$. Initially, set $\mathbf{x}$ to be the empty string. During this backtracking procedure, from a cell $s[i, j]$ choose the maximum among $s[i-1, j-1]$ if $v_i = w_j$, $s[i-1, j]$ and

$s[i, j-1]$. If $s[i-1][j-1]$ is picked, put $v_i = w_j$ in front of the current string $\mathbf{x}$. Recurse on the picked coordinate with the maximum value. Recursion terminates when $(i, j) = (0, 0)$.

d. What is the running time of the DP algorithm in (c)? Briefly motivate your answer. [5 points]

Answer: $O(mn)$. The running time is dominated by the two nested for loops running from 1 to m and 1 to n.

4. **Hamming Distance** [10 points]

Consider two strings $\mathbf{v} = v_1, \ldots, v_n, \mathbf{w} = w_1, \ldots, v_n \in \Sigma^n$ with identical lengths n. The Hamming distance $h(\mathbf{v}, \mathbf{w})$ is the number of positions $i \in [n]$ where v_i differs from w_i. For example, $h(\text{test}, \text{best}) = 1$ since these two strings only differ at the first position, i.e. $v_1 = \text{t} \neq \text{b} = w_1$. Prove that h is a distance function/metric on the space of all strings Σ^n of length n.

Hint: Base your proof on the proof for edit distance given in lecture 3.

Answer:

(a) Non-negativity:
By definition the number of positions at which two strings of length n differ is at least 0 and at most n. Hence, $h(\mathbf{v}, \mathbf{w}) \geq 0$ for any two strings $\mathbf{v}, \mathbf{w} \in \Sigma^n$.

(b) Identity of indiscernibles:
($\Rightarrow$) Let $\mathbf{v}, \mathbf{w} \in \Sigma^n$ with $h(\mathbf{v}, \mathbf{w}) = 0$. Thus, the number of different positions is 0. Hence $\mathbf{v} = \mathbf{w}$. ($\Leftarrow$) Let $\mathbf{v}, \mathbf{w} \in \Sigma^n$ such that $\mathbf{v} = \mathbf{w}$. Thus, the number of positions at which these two strings differ is 0. Hence, $h(\mathbf{v}, \mathbf{w}) = 0$.

(c) Symmetry:
Let $\mathbf{v}, \mathbf{w} \in \Sigma^n$. Clearly, $v_i = w_i$ if and only if $w_i = v_i$. Hence, $h(\mathbf{v}, \mathbf{w}) = h(\mathbf{w}, \mathbf{v})$.

(d) Triangle inequality:
Let $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \Sigma^n$. Assume for a contradiction that $h(\mathbf{v}, \mathbf{w}) > h(\mathbf{v}, \mathbf{u}) + h(\mathbf{u}, \mathbf{w})$. This means that there is a position $i \in [n]$ such that $v_i \neq w_i$, $v_i = u_i$ and $u_i = w_i$. However, if $v_i = u_i$ and $u_i = w_i$ then, by transitivity, $v_i = w_i$. This gives rise to a contradiction. Hence, $h(\mathbf{v}, \mathbf{w}) \leq h(\mathbf{v}, \mathbf{u}) + h(\mathbf{u}, \mathbf{w})$.

5. **Bonus: Total Number of Global Alignments** [20 points]

In this bonus question, we are going to determine the total number of *global alignments* that exist given two strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$. We will assume without loss of generality that $m \leq n$. Recall the matrix representation of an alignment. This is a $2 \times k$ matrix where $k \in \{\max\{m,n\}, \ldots, m+n\}$ such that there is no column with two gaps. Thus, the number k of columns varies from $\max\{m,n\}$ to $m+n$.

a. Explain why, in general (i.e. no prior knowledge on how m and n are related), the number k of columns varies from $\max\{m,n\}$ to $m+n$. [5 points]

Hint: What is the smallest alignment and what is the largest alignment?

Answer: This is very easy to show by contradiction. Assume there exists an alignment with $k < \max\{m,n\}$ columns. We distinguish two cases. First, $\max\{m,n\} = m$. This means that one of the characters of string $\mathbf{v}$ is missing from the alignment.
This violates the requirement of a *global* alignment. Second, $\max\{m,n\} = n$. In this case, one of the characters of string $\mathbf{w}$ is missing from the alignment, thus also violating the requirement of a global alignment. Both cases yield a contradiction. Thus, $k \geq \max\{m,n\}$.
Now assume, again for a contradiction, there exists an alignment with $k > m+n$ columns. Since string $\mathbf{v}$ has only m characters and string $\mathbf{w}$ has only n characters, this alignment must contain a column with two gap characters. This violates the requirement that no alignment contains a column with -/-. Thus, $k \leq m+n$. Hence, $k \in \{\max\{m,n\}, \ldots, m+n\}$.

b. Explain why $k \in \{n, \ldots, m+n\}$ for the case where $m \leq n$. [1 point]

Answer: If $m \leq n$ then $\max\{m,n\} = n$. Hence, $k \in \{n, \ldots, m+n\}$.

c. Suppose that the alignment has length $k \geq n$. In how many different ways can we insert $k-n$ gaps in the second sequence $\mathbf{w}$, yielding gapped sequence $\mathbf{w}'$? [4 points]

Hint: Observe that $\mathbf{w}'$ has length k.

Answer: $\binom{k}{k-n} = \binom{k}{n}$

d. Let $\mathbf{w}'$ be a gapped sequence of length $k \in \{n, \ldots, m+n\}$ such that removing the gaps yields the original sequence $\mathbf{w}$. In how many ways can we insert gaps in $\mathbf{v}$ to obtain an alignment with $\mathbf{w}'$ of length k? [4 points]

Hint: Recall that an alignment does not contain columns with two gaps. In how many different ways we can insert gaps in $\mathbf{v}$ subject to this condition?

Answer: Given the constraint on the alignment length, we must insert exactly $k-m$ gaps in $\mathbf{v}$ to obtain the gapped sequence $\mathbf{v}'$. Gaps in $\mathbf{v}'$ must correspond to non-gaps in $\mathbf{w}'$. Now, the number of non-gap positions in $\mathbf{w}'$ is n. Hence, the number of ways to generate $\mathbf{v}'$ is $\binom{n}{k-m}$.

e. How many alignments of $\mathbf{v}$ and $\mathbf{w}$ are there of a given length $k \in \{n, \ldots, m+n\}$? How many alignments are there of any length? [3 points]

Hint: Combine your answers to the previous two questions.

Answer: Per the previous two answers, the number of alignments of a given length $k \in \{n, \ldots, m+n\}$ is $\binom{k}{n}\binom{n}{k-m}$. To see why, observe that an alignment corresponds to two gapped sequences $\mathbf{v}'$ and $\mathbf{w}'$ obtained from, respectively, $\mathbf{v}$ and $\mathbf{w}$ subject to the constraint that there is no column $-/-$. In question (c), we computed the number of gapped sequences $\mathbf{w}'$ of $\mathbf{w}$ as $\binom{k}{n}$. In question (d), we saw that the number of matching gapped sequences $\mathbf{v}'$ subject to the $-/-$ constraint is $\binom{n}{k-m}$. Combining these two results yields the answer.
The number of alignments of any length is simply

$$\sum_{k=n}^{m+n} \binom{k}{n}\binom{n}{k-m}. \tag{2}$$

f. Give an example of a scoring function $\delta : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$ such that the number of optimal global alignments equals your answer to the previous question. [3 points]

Hint: Think of a border case.

Answer: Set $\delta(c, d) = 0$ for any $(c, d) \in (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\})$. Any alignment will have score 0. Thus all alignments are optimal.