CS466: Introduction to Bioinformatics **Name: Nivedita Rethnakar**

Problem Set 2

Handed out: September 27, 2024 *Due: October 5, 2024*

Instructions: This homework assignment consists of three questions worth a total of 50 points. In addition, there is a bonus question worth an additional 5 points. These questions are based on the material covered in Lectures 6 to 9. **Do not forget to write your name at the top!**

1. **Subquadratic Time Alignment** [15+5 points]

   a. In class we learned how to solve the block alignment problem in time $O(n^2/\log n)$ using the Four Russians Technique. Specifically, in the case of an alphabet of $|\Sigma| = 4$ letters, we pre-computed *all* pairwise alignments of *all* strings of length $t = \log_2(n)/4$. Protein sequences have an alphabet of $|\Sigma| = 20$ letters. How should we choose length t to achieve the time bound of $O(n^2/\log n)$ if $|\Sigma| = 20$? Motivate your answer. [5 points]

For $s = 20$, the number of possible alignments between two blocks is:

$$(s^t)^2 = s^{2t}$$

Set $s^{2t} = n$ to manage precomputation time.
Taking log of both sides (base 2):

$$\log_2(20^{2t}) = \log_2 n$$

$$2t \cdot \log_2 20 = \log_2 n$$

$$t = \frac{\log_2 n}{2 \log_2 20}$$

$$t = \frac{1}{2} \log_{20} n$$

And this ensures a time complexity of $O(n^2/\log n)$

b. How should we choose length t if the size $k = |\Sigma|$ of the alphabet is **not constant**? What is the running time? Motivate your answer. [5 points]
Bonus: +5 points if you explain for what values of k this technique leads to: (i) an asymptotic speedup, (ii) the same running time as regular dynamic programming and (iii) an asymptotic slowdown.

For an alphabet of size $k = |\Sigma|$, the number of possible unique blocks of length t is k^t, and the number of possible block alignments is k^{2t}. To keep the precomputation time manageable $k^{2t} = O(n)$ so the total number of precomputed block alignments is proportional to n, making it feasible.
Let $k^{2t} = n$. Taking log of both sides (base 2) so

$$2t \cdot \log_2 k = \log_2 n$$

$$t = \frac{\log_2 n}{2 \log_2 k}$$

$$t = \frac{1}{2} \log_k n$$

Which is the optimal block length. The dynamic programming matrix is partitioned into $\frac{n}{t} * \frac{n}{t} = \frac{n^2}{t^2}$. For a running time of $O\left(\frac{n^2}{t^2}\right)$ substituting $t = \frac{1}{2}\log_k n$:

$$O\left(\frac{n^2}{\left(\frac{1}{2}\log_k n\right)^2}\right) = O\left(\frac{4n^2}{(\log_k n)^2}\right)$$

So the total running time is $O\left(\frac{4n^2}{(\log_k n)^2}\right)$

Bonus:
When k is small relative to n (ex. $\log_2 k = o(\log_2 n)$): The algorithm achieves an asymptotic speedup over the standard $O(n^2)$ dynamic programming

When k grows proportionally to n (ex $\log_2 k = \Theta(\log_2 n)$): The running time remains $O(n^2)$, similar to standard dynamic programming

When k grows faster than n (e.g., $\log_2 k = \omega(\log_2 n)$): The algorithm experiences an asymptotic slowdown, with the running time exceeding $O(n^2)$

Essentially smaller alphabet sizes lead to speedup, while larger ones can result in slower computation times. By selecting t appropriately, the performance of the algorithm can be based on the size of the alphabet.

c. In their STOC 2015 paper, Backurs and Indyk proved that the edit distance problem cannot be solved in time $O(n^{2-\epsilon})$ where $\epsilon > 0$ under the Strong Exponential Time Hypothesis. To show that the same result also holds for pairwise global sequence alignment, choose an appropriate scoring function $\delta : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$ such that an optimal edit distance alignment is also an optimal global sequence alignment. [5 points]

The scoring function $\delta : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$ would be defined as

$$\delta(a,b) = \begin{cases} 0 & \text{if } a = b \\ 1 & \text{if } a \neq b, \text{ for } a, b \in \Sigma \\ 1 & \text{if } a = \{-\} \text{ or } b = \{-\} \end{cases}$$

- $\delta(a,a) = 0$, for all $a \in \Sigma$ for **matches** showing that aligning the same character from both sequences incurs **no cost**

- $\delta(a,b) = 1$, for all $a, b \in \Sigma,\ a \neq b$ for **mismatches** showing that aligning two different characters incurs a **cost of 1**

- $\delta(a,-) = \delta(-,a) = 1$, for all $a \in \Sigma$ for **gap (insertion or deletion)** where introducing a gap also incurs a **cost of 1**

2. **Carrillo-Lipman** [10 points]

We consider the WEIGHTED SP-EDIT DISTANCE problem, where we are given sequences $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \Sigma^*$ each with length n and a scoring function $\delta : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$. The task is to find a multiple alignment A such that $\mathrm{SP}(A)$ is minimum. We use the Carrillo-Lipman algorithm. Let $\mathbf{v}_{i,j}$ denote the prefix $v_{i,1} \ldots v_{i,j}$ of sequence $\mathbf{v}_i$ of length j. Briefly, $D(i_1, \ldots, i_k)$ denotes the minimum cost of aligning the k prefixes $\mathbf{v}_{1,i_1}, \ldots, \mathbf{v}_{k,i_k}$. On the other hand, $D^+_{a,b}(i, j)$ denotes the minimum cost of the pairwise alignment of suffixes $\mathbf{v}_{a,i}$ and $\mathbf{v}_{b,j}$. In Lecture 7, we considered the $k = 3$ case. We learned that given a heuristic solution with cost z, we know that the optimal alignment does *not* pass through vertex (i_1, i_2, i_3) if

$$D(i_1, i_2, i_3) + D^+_{1,2}(i_1, i_2) + D^+_{1,3}(i_1, i_3) + D^+_{2,3}(i_2, i_3) > z. \tag{1}$$

a. Consider the general case with $k \in \mathbb{N}$ sequences. Let $(i_1, \ldots, i_k) \in [n]^k$ and let $D(i_1, \ldots, i_k)$ be the optimal cost for aligning prefixes $\mathbf{v}_{1,i_1}, \ldots, \mathbf{v}_{k,i_k}$. Let z be the cost of an alignment of $\mathbf{v}_1, \ldots, \mathbf{v}_k$. Under which condition do we know that the optimal alignment does *not* pass through vertex $(i_1, \ldots, i_k)$? [5 points]
Hint: Update Equation (1).

The optimal alignment does not pass through vertex $(i_1, i_2, \ldots, i_k)$ if:

$$D(i_1, i_2, \ldots, i_k) + \sum_{1 \le a < b \le k} D^+_{a,b}(i_a, i_b) > z.$$

This represents a lower bound on the cost of the alignment passing through $(i_1, i_2, \ldots, i_k)$. If this lower bound exceeds z, no optimal alignment passes through this vertex, allowing us to prune it

b. Consider the SP-GLOBAL ALIGNMENT problem, where we have the same input $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \Sigma^*$ but aim to find an alignment A with *maximum* score $\mathrm{SP}(A)$. Let z be the score of an alignment of $\mathbf{v}_1, \ldots, \mathbf{v}_k$. Under which condition do we know that the optimal alignment does *not* pass through vertex $(i_1, \ldots, i_k)$? [5 points]

The optimal alignment does not pass through vertex $(i_1, i_2, \ldots, i_k)$ if:

$$D(i_1, i_2, \ldots, i_k) + \sum_{1 \le a < b \le k} D^+_{a,b}(i_a, i_b) < z.$$

This upper bound on the maximum score through the vertex helps us prune the vertex if it cannot exceed the known score z

3. **Tree and Center Star Alignment** [25 points]

   Consider the following four strings $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$:

   $\mathbf{v}_1$: CGGGAGTGA
   $\mathbf{v}_2$: CGTTAGGGA
   $\mathbf{v}_3$: CGTTGGA
   $\mathbf{v}_4$: AGTTGGGAA

   Here are their pre-computed optimal pairwise alignments with a cost of 0 for matches and 1 otherwise:

$\mathbf{v}_1$: CGGGAGTGA $\mathbf{v}_2$: CGTTAGGGA $D(\mathbf{v}_1, \mathbf{v}_2) = 3$	$\mathbf{v}_1$: CGGGAGTGA $\mathbf{v}_3$: C-GTTG-GA $D(\mathbf{v}_1, \mathbf{v}_3) = 4$	$\mathbf{v}_1$: CG--GGAGTGA $\mathbf{v}_4$: AGTTGG-G-AA $D(\mathbf{v}_1, \mathbf{v}_4) = 6$
$\mathbf{v}_2$: CGTTAGGGA $\mathbf{v}_3$: CGTT--GGA $D(\mathbf{v}_2, \mathbf{v}_3)$: 2	$\mathbf{v}_2$: CGTTAGGG-A $\mathbf{v}_4$: AGTT-GGGAA $D(\mathbf{v}_2, \mathbf{v}_4) = 3$	$\mathbf{v}_3$: CGTT-GG-A $\mathbf{v}_4$: AGTTGGGAA $D(\mathbf{v}_3, \mathbf{v}_4) = 3$

a. Give (i) all **induced pairwise alignments** and (ii) **their scores** of the following multiple sequence alignment of $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$.

$\mathbf{v}_1$: CGGGAGTGA-
$\mathbf{v}_2$: CGTTAGGGA-
$\mathbf{v}_3$: CGTT-GG-A-
$\mathbf{v}_4$: AGTT-GGGAA

Use the same cost/scoring function as EDIT DISTANCE. So a cost of 0 for matches and 1 for mismatches/gaps. [5 points]

(i) All induced pairwise alignments:

1. D(v1, v2):
$\mathbf{v}_1$: CGGGAGTGA-
$\mathbf{v}_2$: CGTTAGGGA-
Cost: 3

2. Pair (v1, v3):
$\mathbf{v}_1$: CGGGAGTGA-
$\mathbf{v}_3$: CGTT-GG-A-
Cost: 5

3. Pair (v1, v4):
$\mathbf{v}_1$: CGGGAGTGA-
$\mathbf{v}_4$: AGTT-GGGAA
Cost: 6

4. Pair (v2, v3):
$\mathbf{v}_2$: CGTTAGGGA-
$\mathbf{v}_3$: CGTT-GG-A-
Cost: 2

5. Pair (v2, v4):
$\mathbf{v}_2$: CGTTAGGGA-
$\mathbf{v}_4$: AGTT-GGGAA
Cost: 3

6. Pair (v3, v4):
$\mathbf{v}_3$: CGTT-GG-A-
$\mathbf{v}_4$: AGTT-GGGAA
Cost: 3

(ii) Scores:

- D(v1, v2): 3
- D(v1, v3): 5
- D(v1, v4): 6
- D(v2, v3): 2
- D(v2, v4): 3
- D(v3, v4): 3

b. Find an alignment of $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$ **consistent with** the tree $\mathbf{v}_2 - \mathbf{v}_3 - \mathbf{v}_4 - \mathbf{v}_1$. Present your answer by (i) indicating which two sequences you are aligning at each step, and (ii) provide all intermediate multiple alignments as well as the final multiple alignment of the strings. [10 points]

Aligning $\mathbf{v}_2$ and $\mathbf{v}_3$
$\mathbf{v}_2$: C G T T A G G G A
$\mathbf{v}_3$: C G T T G G A

Alignment:
$\mathbf{v}_2$: C G T T A G G G A
$\mathbf{v}_3$: C G T T - - G G A

Cost: 2

Aligning with $\mathbf{v}_4$
$\mathbf{v}_4$: A G T T G G G A A

Alignment:
$\mathbf{v}_2$: C G T T A G G G A
$\mathbf{v}_3$: C G T T - - G G A
$\mathbf{v}_4$: A G T T G G G A A

Cost between $\mathbf{v}_4$ and $\mathbf{v}_2$: 3
Cost between $\mathbf{v}_4$ and $\mathbf{v}_3$: 4

Total cost for $\mathbf{v}_4$: 7

Aligning with $\mathbf{v}_1$ $\mathbf{v}_1$: C G G G A G T G A
Final Alignment:
$\mathbf{v}_1$: C G G G A G T G A
$\mathbf{v}_2$: C G T T A G G G A
$\mathbf{v}_3$: C G T T - - G G A
$\mathbf{v}_4$: A G T T G G G A A

Cost between $\mathbf{v}_1$ and $\mathbf{v}_2$: 3
Cost between $\mathbf{v}_1$ and $\mathbf{v}_3$: 5
Cost between $\mathbf{v}_1$ and $\mathbf{v}_4$: 6

Total cost for $\mathbf{v}_1$: 14

Summary of Costs: 23
- Aligning $\mathbf{v}_2$ and $\mathbf{v}_3$: 2
- Aligning $\mathbf{v}_4$: 7
- Aligning $\mathbf{v}_1$: 14

Total Alignment Cost: 23

c. Create a star alignment of the four strings s_1, s_2, s_3, s_4 using the pre-computed optimal pairwise alignments provided above. Present your answer by (i) identifying the center sequence s_c, (ii) the quantity $\sum_{i=1}^{4} d(s_c, s_i)$ and (iii) the final multiple alignment of the strings. **Show your work.** Include all intermediate multiple alignments that you generate. [10 points]

(i) Identifying the Center Sequence s_c: Total Distances:
$D(v_1) = 3 + 5 + 6 = 14$
$D(v_2) = 3 + 2 + 3 = 8$
$D(v_3) = 5 + 2 + 3 = 10$
$D(v_4) = 6 + 3 + 3 = 12$
The center sequence s_c is v_2 since it has the smallest total distance.

(ii) Calculating $\sum_{i=1}^{4} d(s_c, s_i)$:

$$\sum_{i=1}^{4} d(v_2, s_i) = d(v_2, v_1) + d(v_2, v_2) + d(v_2, v_3) + d(v_2, v_4) = 3 + 0 + 2 + 3 = 8$$

(iii) Final Multiple Alignment:
Aligning v_2 and v_1
v_2: C G T T A G G G A -
v_1: C G G G A G T G A -
Total cost: 3

Aligning v_2 and v_3
v_2: C G T T A G G G A -
v_3: C G T T - - G G A -
Total cost: 2

Aligning v_2 and v_4
v_2: C G T T A G G G A -
v_4: A G T T - G G G A A
Total cost: 3

Final Multiple Alignment:
v_1: C G G G A G T G A -
v_2: C G T T A G G G A -
v_3: C G T T - - G G A -
v_4: A G T T - G G G A A

Summary of Costs:
$d(v_2, v_1) = 3$
$d(v_2, v_3) = 2$
$d(v_2, v_4) = 3$
Total cost: $\sum_{i=1}^{4} d(v_2, s_i) = 8$