9.3 Protein Comparison in Bioinformatics

Question 1

Theorem 1. *For all $i, j > 0, i \leq n, j \leq m$*

$$D(i,j) = \min D(i-1,j)+1, D(i,j-1)+1, D(i-1,j-1)+s(S_i,T_j)$$

where

$$s(a,b) = \mathbb{1}[a=b]$$

Further, the boundary conditions to use are:

$$D(0,0) = 0, D(i,0) = i, D(0,j) = j$$

Proof. There are four possible edits: R,I,M,D. The claim is clearly true for $(i,j) \in \{(0,1),(1,0),(1,1)\}$ Further, we use the given boundary conditions (the reasoning for $(0,0)$ is quite straightforward. As for $(0,j)$ (and thus $(i,0)$) the reasoning is that we would require j inserts to get the appropriate string when starting from an empty string.

The most important thing to prove is the optimal substructure property. Since edits are done locally (as they do not directly depend on what the other possible characters are) suggests that problem has the optimal substructure property. We shall prove it below:

Claim 1. *The problem in question has the optimal substructure property.*

Proof. Consider we start with the string S. Further, consider a function C that, given a sequence of operations $P = P_1P_2P_3\cdots P_k$, outputs the cost of transforming S into another string W using the editing transcript P, that is $d(S,P) = C(P)$.

Consider an editing transcript $P = P_1P_2P_3\cdots P_k$ that transforms S to T. Further, let $P^j = P[1,j]$, and let T^j be the sequence one obtains by applying P^j to S.

We intend to show that if $C(P^i) = d(S,T^i)$, then $C(P^{i-1}) = d(S,T^{i-1})$, as then the claim follows immediately. Suppose not. Then due to the definition of d, we must have $C(P^{i-1}) > d(S,T^{i-1})$. Then we have some sequence P' that transforms S to T^{i-1} with edit length $d(S,T^{i-1})$. Now consider the sequence $P'' = P' \bigcup P_i$. Then

$$\begin{aligned} C(P'') &= d(S,T^{i-1}) + C(P_i) \\ &< C(P_i) = d(S,T^i) \end{aligned}$$

This is absurd, as this contradicts minimality of $d(S,T^i)$!

Now, the optimal substructure is immediate. ■

Since optimal substructure property is true, we can use dynamic programming approach! We use strong induction, and suppose that $D(x,y)$ suggested by the theorem is the optimal edit distance for tuples (x,y) such that $x \leq i, y \leq j$ but $(x,y) \neq (i,j)$.

Now we consider the tuple (i,j). We have a few possible cases, depending on what is the most optimal possible edit at that particular instance:

1. **Match or Replace**: In the case of match, $D(i,j) = D(i-1,j-1)$ as then both new characters contribute zero score. In the case of replace, $D(i,j) = D(i-1,j-1)+1$ for clear reasons.This can be summarised as

$$D(i,j) = D(i-1,j-1) + s(S_i,T_j)$$

2. **Insert**: This instance can be considered the same as the first case above, but with the string S extended by appending T_j at the end. Then the new edit distance is $D(i, j) = 1 + D(i, j-1)$

3. **Delete**: Similar (but instead of appending to S, do similar with S_i and T to the second case, suggesting $D(i, j) = 1 + D(i-1, j)$

This indeed then proves that the claimed recursion is correct. ■

Question 2

The appropriate program can be found in the appendix.

The edit distance between `seashells` and `shesells` are 3 edits, with the proposed edit sequence being `MDMIMIMMMM`.

The complexity of the algorithm used if $O(mn)$ due to a nested loop that runs $(n+1)\cdot(m+1)$ times. To ensure that we are not calculating each value of $D(i, j)$ multiple times, we use memoization and store the values in a 2×2 list. We also use the fact the boundary conditions given in Theorem 1.

Question 3

The modified program can be found in the appendix. The modified program uses pointers to backtrack and obtain an optimal editing transcript.

The edit distance between the two proteins A and B is 83 edits. The first 50 steps of an optimal alignment are detailed in the

`MRRRMRDDDDMMMRMMRMMRMRRRRMRRMMRMMMMRMMMMRDMIMRMMRM`

The first 50 steps are further detailed and listed in the appendix.

Question 4

We use the BLOSUM matrices of Henikoff and Henikoff referenced in the project brief, with Delete and Insert scores of -8. We will consider the problem of maximising $v(S, T)$, the maximum score of all edit transcripts of S and T. This problem is similar to the earlier problem of minimising the edit distance.

Using the same logic (and similar notation) as in Claim 1, we notice that

$$v(i, j) = \max v(i-1, j) - 8, v(i, j-1) - 8, v(i-1, j-1) + s(S_i, T_j)$$

Here $s(S_i, T_j)$ is the (i, j)th entry of the BLOSUM Matrix. This new problem requires new boundary conditions. The most sensible ones are

$$v(0, 0) = 0, v(i, 0) = -8i, v(0, j) = -8j$$

, with the same reasoning as for the boundary conditions in Claim 1.

For the protein sequences A and B, the score $v(A, B)$ is 290. The appropriate edit transcript is:

`MDDRMDDRRRMMMRMMRMMRMRRRRMRRMMRMMMMRMMMMRRRMRMMRMR`

The first 50 steps are further detailed and listed in the appendix.

Question 5

We shall now consider situations where we can insert/delete large sequences of amino acids, say of length l, at once with a certain score, $w(l)$. We are told $w(l) < 0$ for $l \geq 1$.

Let $v_{\text{gap}}(S, T)$ be the gap-weighted score of two strings S and T, and let $V_{\text{gap}}(i, j) = v_{\text{gap}}(S[1, i], T[1, j])$.

We are further given the definitions of $E(i,j), F(i,j)$ and $G(i,j)$, which are as follows:

$$\begin{aligned} E(i,j) &= \max_{0 \le k \le j-1} \{V_{\text{gap}}(i,k) + w(j-k)\} \\ F(i,j) &= \max_{0 \le k \le i-1} \{V_{\text{gap}}(k,j) + w(i-k)\} \\ G(i,j) &= V_{\text{gap}}(i-1,j-1) + s(S_i, T_j) \\ V_{\text{gap}}(i,j) &= \max\{E(i,j), F(i,j), G(i,j)\} \end{aligned}$$

We now consider the situation where $w(l) = u$ for all $l \ge 1$, where $u \le 0$ is a constant. In this case:

$$\begin{aligned} E(i,j) &= \max_{0 \le k \le j-1} \{V_{\text{gap}}(i,k) + u\} \\ F(i,j) &= \max_{0 \le k \le i-1} \{V_{\text{gap}}(k,j) + u\} \\ G(i,j) &= V_{\text{gap}}(i-1,j-1) + s(S_i, T_j) \\ V_{\text{gap}}(i,j) &= \max\{E(i,j), F(i,j), G(i,j)\} \end{aligned}$$

There exists a $O(mn)$ algorithms to implement this case. This algorithm is implemented in the appendix.

We now describe the details of the algorithm. Notice that

$$\begin{aligned} E(i,j) &= \max\{\max_{0 \le k \le j-2} V_{\text{gap}}(i,k) + u, V_{\text{gap}}(i,j-1) + u\} \\ &= \max\{E(i,j-1), V_{\text{gap}}(i,j-1) + u\} \end{aligned}$$

Similarly,

$$F(i,j) = \max\{F(i-1,j), V(i-1,j) + u\}$$

These observations imply that we would require only a single nested loop that runs $m \times n$ times. In fact, we initialise a large array of $(m+1) \times (n+1)$ entries, and use the boundary conditions described below to initialise the array for tuples of the form $(i,0)$ and $(0,j)$. This is done in $O(\max\{m,n\})$ iterations. Then we use a single nested loop that runs mn times for $i = 1, \cdots, m$, and $j = 1, \cdots, n$. During each iteration we can calculate $E(i,j), F(i,j), G(i,j)$ in $O(1)$ time (due to the recursions given above!), and thus $V_{\text{gap}}(i,j)$ in $O(1)$ time. This implies that the overall time complexity is $O(mn)$.

We now require new boundary conditions as well. One might notice that we require boundary conditions for V_{gap} as well as E, F and G!

For V, the boundary conditions are the familiar one. For all $i, j > 0$:

$$V(0,0) = 0, V(0,j) = V(i,0) = u$$

For G, the boundary conditions are again straightforward. For all $i, j > 0$:

$$G(i,0) = G(0,j) = G(0,0) = 0$$

The reasoning for these boundary conditions is that with 0 characters in at least one of the strings, there is nothing to Match/Replace!

For E, we select the following boundary conditions. For $i, j > 0$:

$$E(0,0) = 0, E(0,j) = u, E(i,0) = 2u$$

The first boundary condition is clear. The second boundary condition follows from the fact that can insert all j characters at once! The third boundary condition is interesting, however. Note

$$\begin{aligned} E(i,1) &= \max\{E(i,0), V(i,0) + u\} \\ &= \max\{E(i,0), 2u\} \end{aligned}$$

So we in fact require that $E(i,0) \le 2u$ for the correct value of $E(i,1)$. Notice, curiously, that $E(i,0) = 2u$ makes little sense if we use the overall interpretation of what $E(i,j)$ means, as it makes no sense to be able to insert something into a non-empty string and end up with an empty string!

Similarly, we get boundary conditions for F. For $i, j > 0$:

$$F(0,0) = 0, F(i,0) = u, E(0,j) = 2u$$

Question 6

The implementation for this question can be found in the appendix. We now use the BLOSUM matrix for scoring function s, and set $u = -12$. For proteins C and D, we get that $v_{\text{gap}}(C, D) = 1236$. The appropriate edit transcript is:

`MDDDDDDDDDDDDDDDDDDDDDDMRRRMRRRRRMRRDRMRRRRMRRMMRDD`

The first 50 steps are further detailed and listed in the appendix.

Question 7

Consider the new problem for simplicity, as detailed in the project brief. We now are concerned with only two amino acids - a, b. Let $U^n = U_1^n \cdots U_n^n$ be a random protein of length n, where U_i^n are iid (independent and identically distributed) random variables, that can take values from the set $\{a, b\}$ such that $\mathbb{P}(U_i^n = a) = p$ and $\mathbb{P}(U_i^n = b) = 1 - p = q$. Further, our scoring matrix is such that $s(a, a) = s(b, b) = 1$ and $s(a, b) = s(b, a) = -1$.

Consider now two random iid proteins U^n and V^n.

Theorem 2. *Consider the above scenario. Further, consider $w(l) = u$ for all $l \geq 1$, where $u < 0$. Then for all $0 \leq p \leq 1$, and for all $u \leq 0$:*

$$\liminf_{n \to \infty} \frac{\mathbb{E}\left(v_{gap}(U^n, V^n)\right)}{n} > 0$$

Proof. First note:

$$\begin{aligned}
\mathbb{E}\left[v_{\text{gap}}(U^1, V^1)\right] &= p^2 s(a, a) + q^2 s(b, b) + 2pq(v_{\text{gap}}(a, b)) \\
&= p^2 + q^2 + 2pq \max\{-1, 2u\} \\
&\geq (q - p)^2 \\
&= (2p - 1)^2 = \mathbb{E}\left[s(U^1, V^1)\right]
\end{aligned}$$

Then, for all $n \geq 1$,

$$\begin{aligned}
\mathbb{E}\left[v_{\text{gap}}(U^n, V^n)\right] &= \mathbb{E}\left[G(n, n), E(n, n), F(n, n)\right] \\
&\geq \mathbb{E}\left[G(n, n)\right] \\
&= \mathbb{E}\left[v_{\text{gap}}(U^{n-1}, V^{n-1})\right] + (2p - 1)^2 \\
&\geq n \cdot (2p - 1)^2
\end{aligned}$$

Now define function f such that $f(n) = \mathbb{E}\left[v_{\text{gap}}(U^n, V^n)\right]$. We have then shown that, for all $n \geq 1$,

$$\frac{f(n)}{n} \geq (2p - 1)^2$$

. Remarkably, we are then done and the claim follows if $p \neq \frac{1}{2}$.

What is $p = \frac{1}{2}$? In fact, we will prove something stronger than above, and aim to improve the above bound (which involved only replacement/matching edits to get the bound). This would then imply the case of $p = \frac{1}{2}$ (in fact it implies the case for general p!).

If $n = 1 - 2u$ and all the amino acids are not matched, then purely replace/match edits lead to a score $2u - 1$ (attained by all replacements), while pure deletions and insertions lead to a score of $2u$. This implies that in this case we can increase our lower bound. So in the case of general n, we can increase our lower bound by considering dividing our string into blocks of size $1 - 2u$ and using pure deletion/insertion in these blocks if there are no matches, instead of pure replace/match edits. As a result :

$$\begin{aligned}
f(n) &\geq n \cdot (2p - 1)^2 + 1 \times \left\lfloor \frac{n}{1 - 2u} \right\rfloor * \mathbb{P}(1 - 2u \text{ string length, with no matched letter}) \\
&\geq n \cdot (2p - 1)^2 + \left(\frac{n}{1 - 2u} - 1\right) (2pq)^{1 - 2u}
\end{aligned}$$

This immediately implies that (not only for $p = \frac{1}{2}$, but for general $p \in [0, 1]$:

$$\begin{aligned}\liminf_n \frac{f(n)}{n} &\geq \liminf_n \left((2p-1)^2 + \left(\frac{1}{1-2u} - \frac{1}{n} \right) (2pq)^{1-2u} \right) \\ &= (2p-1)^2 + \frac{1}{1-2u}(2pq)^{1-2u} > 0\end{aligned}$$

■

We will now claim something stronger

Theorem 3. *Using the notation as in Theorem 2, the limit below exists and the following is true:*

$$\lim_{n \to \infty} \frac{\mathbb{E}\left(v_{gap}(U^n, V^n)\right)}{n} > 0$$

Proof. We will use the notation we established earlier in proof of Theorem 2.

We note that $f(n+1) \geq f(n) + (2p-1)^2 \geq f(n)$, ie $\{f(n)\}$ is a monotonically increasing sequence. Also, it is quite clear that $f(n) \leq n$ (which is attained in the case of all n matches). So might conclude that:

$$f(n+1) = f(n) + (2p-1)^2 + g(n)$$

where $g(n)$ is some supplemental function that we are unaware of, and $g(n) \geq 0$ by above.

We in fact notice that $g(n)$ must be monotonically increasing as well, as $g(n)$ is the expected score improvement if we consider more complicated Delete and Insert edit sequences when U^{n-1}, V^{n-1} interact with the n^{th} character of U, V. Then, $g(n) \geq g(n-1)$ as we either get the same increase or do something new which should still increase our overall score.

So now we have $\frac{f(n)}{n} \leq 1$ and $\{f(n)\}, \{g(n)\}$ are increasing sequences, with $g(n) \geq 0$ for all n. Suppose $g(n)$ is bounded above. Then as it is a monotonically increasing sequence, $\lim_n g(n) = G$ exists. Let further $\frac{f(n)}{n} = h(n)$. Notice $G \geq 0$ necessarily, as $g(n) \geq 0$. Then, notice

1. $\underline{G > 1}$:

$$\begin{aligned}n(h(n) - h(n-1)) &= \left(f(n-1) + (2p-1)^2 + g(n) - \frac{nf(n-1)}{n-1} \right) \\ &= \frac{-f(n-1)}{n-1} + (2p-1)^2 + g(n) \\ &\geq g(n) - 1\end{aligned}$$

   So if $G > 1$, then this implies that eventually, $h(n)$ is monotonically increasing. Since it is also bounded above by 1, the limit of $h(n)$ thus exists.

2. $\underline{G \leq 1}$: In this case, by above, we can conclude that eventually $h(n)$ is monotonically decreasing. Since $h(n)$ is bounded below by bounds we found in the proof of Theorem 2, we conclude that the limit of $h(n)$ exists!

Now suppose that $g(n)$ is unbounded above. Then we notice that eventually $g(n) > 1$ for all $n > N$ for some N. But then, using the argument above for the case of $G > 1$, we conclude that $h(n)$ is monotonically increasing and thus the limit of $h(n)$ exists.

Thus we have proven that the limit $\frac{f(n)}{n}$ exists. As for it being positive, that follows from the fact that $\liminf_n \frac{f(n)}{n} > 0$ as was shown in Theorem 2 (and using standard properties of $\liminf$ and $\lim$). ■

Question 8

Consider the case of $u = -3, p = \frac{1}{2}$. The program to estimate $\frac{1}{n}\mathbb{E}[v_{\text{gap}}(U^n, V^n)]$ can be found in the appendix. We estimate by randomly generating the protein sequences U^n, V^n and calculating $\frac{1}{n} v_{\text{gap}}(U^n, V^n)$. We do this multiple times, with M trials, and take the average to get the estimate.

The tabulated values for these trials with $M = 500$ can be found below in Table 1.

n	Estimate of $\frac{1}{n}\mathbb{E}[v_{\text{gap}}(U^n, V^n)]$
10	0.092
100	0.37112
1000	0.42913799999999996
2000	0.438875
5000	0.4393333333

Table 1: Estimates of $\frac{1}{n}\mathbb{E}[v_{\text{gap}}(U^n, V^n)]$ for various values of n

We notice from above that with each increase by a factor of 10, the estimated value increases by $\frac{1}{5}\times$ previous increase. Further, we have decreased our values of M from 500 earlier for larger values of n as the variance decreases and becomes negligible as $n \to \infty$. Thus, we can naively estimate using a geometric series that

$$\frac{1}{n}\mathbb{E}[v_{\text{gap}}(U^n, V^n)] = 0.092 + 0.27912 * \sum_{n=1}^{\infty}(1/5)^n \approx 0.4409$$

Question 9

We now aim to find a pair of substrings S', T' of S, T (notated as $S' \leq S,\ T' \leq T$ from now) with highest alignment score,

$$v_{\text{sub}}(S, T) = \max\{v(S', T') \mid S' \leq S,\ T' \leq T\}$$

For the remainder of this project, we shall write $s(-, a) = s(a, -) < 0$ for the score of insertion or deletion.

We also define

$$v_{\text{sfx}}(S, T) = \max\{v(S', T') \mid S' \text{ suffix of } S,\ T' \text{ suffix of } T\}$$

Now we show an important result.

Theorem 4.

$$v_{sub}(S, T) = \max\{v_{sfx}(S', T') \mid S' \text{ prefix of } S,\ T' \text{ prefix of } T\}$$

Proof.

$$\begin{aligned}
\text{RHS} &= \max_{\substack{0\leq i\leq m,\\ 0\leq j\leq n}}\{v_{\text{sfx}}(S[1,i], T[1,j])\} \\
&= \max_{\substack{0\leq i\leq m,\\ 0\leq j\leq n}}\ \max_{\substack{0\leq a\leq i,\\ 0\leq b\leq j}}\{v(S[a,i], T[b,j])]\} \\
&= \max\{v(S', T') \mid S' \leq S,\ T' \leq T\} = v_{\text{sub}}(S, T)
\end{aligned}$$

where the third equality follows from the facts that m, n are finite and that one can represent any substring $S' \leq S$ (and similarly for T' and T) as $S' = S[i_1, j_1]$ for two indices i_1, j_1 with $j_1 \geq i_1$. ■

Question 10

Let $V_{\text{sfx}}(i, j) = v_{\text{sfx}}(S[1, i], T[1, j])$. We aim to prove another important result:

Theorem 5.

$$V_{sfx}(i, j) = \max\begin{cases} 0, \\ V_{sfx}(i-1, j-1) + s(S_i, T_j), \\ V_{sfx}(i-1, j) + s(S_i, -), \\ V_{sfx}(i, j-1) + s(-, T_j) \end{cases}$$

with boundary conditions $V_{sfx}(i, 0) = V_{sfx}(0, j) = 0$.

Proof. By analogous proof to that for Claim 1 in Theorem 1, this problem **also** has the optimal substructure property, and thus we can use dynamic programming!

Note that

$$V_{\text{sfx}}(i,j) = \max_{\substack{0\leq k\leq i,\\ 0\leq l\leq j}} \{v(S[k,i],T[l,j])\}$$

Suppose that $V_{\text{sfx}}(i,j)$ is attained by the substrings $S[\tilde{k},i]$ and $T[\tilde{l},j]$. Then they could be empty, in which case the $V_{\text{sfx}}(i,j)=0$. Otherwise, they are non-empty. Then by Theorem 1

$$v(S[\tilde{k},i],T[\tilde{l},j]) = \max\begin{cases} v(S[\tilde{k},i-1],T[\tilde{l},j-1]) + s(S_i,T_j), \\ v(S[\tilde{k},i-1],T[\tilde{l},j]) + s(S_i,-), \\ v(S[\tilde{k},i],T[\tilde{l},j-1]) + s(-,T_j) \end{cases}$$

Now further note that $S[\tilde{k},i]$ and $T[\tilde{l},j-1]$ are the optimal strings to attain $V_{\text{sfx}}(i,j-1)$. If not, then the ideal substrings have at least $S[\tilde{k}-1]$ and/or $T[\tilde{l}-1]$. Regardless, this contradicts the maximality of $S[\tilde{k},i]$ and $T[\tilde{l},j]$ for (i,j) as $S[\tilde{k}-1,i]$ and $T[\tilde{l}-1,j-1]$ (or appropriate adjustments) are strictly better! The same conclusion is true for the tuples $(i-1,j)$ and $(i-1,j-1)$. So start of ideal suffixes is unchanged!

The theorem now follows. ■

Question 11

We aim to use the BLOSUM matrix with insertion and deletion scores of -2.

Using Theorem 4 and Theorem 5, we can get an $O(mn)$ algorithm. This is because by these theorems we conclude:

$$v_{\text{sub}}(S,T) = \max_{\substack{0\leq i\leq m,\\ 0\leq j\leq n}} \{V_{\text{sfx}}(i,j)\}.$$

Further, using Theorem 5, we can use memoization to calculate $V_{\text{sfx}}(i,j)$ for all tuples (i,j) in $O(mn)$ running time (by using two loops that use the boundary conditions to initialise a $n\times m$ list $V_{\text{sfx}}(i,j)$, and then further use a nested loop that runs $m\cdot n$ times to calculate $V_{\text{sfx}}(i,j)$ for all (i,j)). This bit of the overall algorithm is similar to algorithms we have seen earlier in this project.

Then we find the maximum over a list of $(m+1)\cdot(n+1)$ list $V_{\text{sfx}}(i,j)$. Thus, the overall time complexity of the algorithm is $O(mn+mn)=O(mn)$.

For proteins C and D, the value of v_{sub} is therefore 1312.

Appendix A: Listings of the first 50 steps of an optimal alignment sequence

Question 3

The first 50 steps of an optimal alignment sequence for this question are below:

```
Step 1 : Match M
Step 2 : Replace G with A
Step 3 : Replace L with D
Step 4 : Replace S with F
Step 5 : Match D
Step 6 : Replace G with A
Step 7 : Delete E
Step 8 : Delete W
Step 9 : Delete Q
Step 10 : Delete L
Step 11 : Match V
Step 12 : Match L
Step 13 : Match K
Step 14 : Replace V with C
Step 15 : Match W
Step 16 : Match G
Step 17 : Replace K with P
Step 18 : Match V
Step 19 : Match E
Step 20 : Replace G with A
Step 21 : Match D
Step 22 : Replace L with Y
Step 23 : Replace P with T
Step 24 : Replace G with T
Step 25 : Replace H with M
Step 26 : Match G
Step 27 : Replace Q with G
Step 28 : Replace E with L
Step 29 : Match V
Step 30 : Match L
Step 31 : Replace I with T
Step 32 : Match R
Step 33 : Match L
Step 34 : Match F
Step 35 : Match K
Step 36 : Replace T with E
Step 37 : Match H
Step 38 : Match P
Step 39 : Match E
Step 40 : Match T
Step 41 : Replace L with Q
Step 42 : Delete E
Step 43 : Match K
Step 44 : Insert L
Step 45 : Match F
Step 46 : Replace D with P
Step 47 : Match K
Step 48 : Match F
Step 49 : Replace K with A
Step 50 : Match G
```

Question 4

The first 50 steps of an optimal alignment sequence for this question are below:

```
Step 1 : Match M
```

```
Step 2 : Delete G
Step 3 : Delete L
Step 4 : Replace S with A
Step 5 : Match D
Step 6 : Delete G
Step 7 : Delete E
Step 8 : Replace W with F
Step 9 : Replace Q with D
Step 10 : Replace L with A
Step 11 : Match V
Step 12 : Match L
Step 13 : Match K
Step 14 : Replace V with C
Step 15 : Match W
Step 16 : Match G
Step 17 : Replace K with P
Step 18 : Match V
Step 19 : Match E
Step 20 : Replace G with A
Step 21 : Match D
Step 22 : Replace L with Y
Step 23 : Replace P with T
Step 24 : Replace G with T
Step 25 : Replace H with M
Step 26 : Match G
Step 27 : Replace Q with G
Step 28 : Replace E with L
Step 29 : Match V
Step 30 : Match L
Step 31 : Replace I with T
Step 32 : Match R
Step 33 : Match L
Step 34 : Match F
Step 35 : Match K
Step 36 : Replace T with E
Step 37 : Match H
Step 38 : Match P
Step 39 : Match E
Step 40 : Match T
Step 41 : Replace L with Q
Step 42 : Replace E with K
Step 43 : Replace K with L
Step 44 : Match F
Step 45 : Replace D with P
Step 46 : Match K
Step 47 : Match F
Step 48 : Replace K with A
Step 49 : Match G
Step 50 : Replace L with I
```

Question 6

The first 50 steps of an optimal alignment sequence for this question are below:

```
Step 1 : Match M
Step 2 : Delete T
Step 3 : Delete S
Step 4 : Delete D
Step 5 : Delete C
Step 6 : Delete S
Step 7 : Delete S
Step 8 : Delete T
```

```
Step 9 : Delete H
Step 10 : Delete C
Step 11 : Delete S
Step 12 : Delete P
Step 13 : Delete E
Step 14 : Delete S
Step 15 : Delete C
Step 16 : Delete G
Step 17 : Delete T
Step 18 : Delete A
Step 19 : Delete S
Step 20 : Delete G
Step 21 : Delete C
Step 22 : Delete A
Step 23 : Match P
Step 24 : Replace A with Y
Step 25 : Replace S with N
Step 26 : Replace S with F
Step 27 : Match C
Step 28 : Replace S with L
Step 29 : Replace V with P
Step 30 : Replace E with S
Step 31 : Replace T with L
Step 32 : Replace A with S
Step 33 : Match C
Step 34 : Replace L with R
Step 35 : Replace P with T
Step 36 : Delete G
Step 37 : Replace T with S
Step 38 : Match C
Step 39 : Replace A with S
Step 40 : Replace T with S
Step 41 : Replace S with R
Step 42 : Replace R with P
Step 43 : Match C
Step 44 : Replace Q with V
Step 45 : Replace T with P
Step 46 : Match P
Step 47 : Match S
Step 48 : Replace F with C
Step 49 : Delete L
Step 50 : Delete S
```

Appendix B: Program Listings

Listing 1: Program for Question 2

```
#==========
#start of def

def s(Si,Tj):
    if(Si==Tj):
        return 0
    else:
        return 1

#end of def
#==========
#start of def

def edit_distance(S,T):
    m=len(S)
    n=len(T)
    D=[[0 for j in range(n+1)]for i in range(m+1)]
    D[0][0]= 0
    for i in range(1,m+1):
        D[i][0]=i
    for j in range(1,n+1):
        D[0][j]=j
    for i in range(1,m+1):
        for j in range(1,n+1):
            D[i][j]=min(min(D[i][j-1]+1,D[i-1][j]+1),s(S[i-1], T[j-1])+D[i
   -1][j-1])
    return D[m][n]
#Complexity of this is O(nm)

#end of def
#==========
```

Listing 2: Program for Question 3

```
#==========
#start of def

def convert_edit_sequence_to_alignment(S,T,pointers):
    m=len(S)
    n=len(T)
    i=m
    j=n
    str1=''
    str2=''
    str_match=''
    while(i>0 or j>0):
        if(pointers[i][j]=='D'):
            str1=S[i-1]+str1
            str2=T[j-1]+str2
            if(S[i-1]==T[j-1]):
                str_match='M'+str_match
            else:
                str_match='R'+str_match
            i-=1
            j-=1
        elif(pointers[i][j]=='L'):
            str1=' '+str1
            str2=T[j-1]+str2
            str_match='I'+str_match
            j-=1
        else:
            str1=S[i-1]+str1
            str2=' '+str2
            str_match='D'+str_match
            i-=1
    print(str_match)
    print(str1)
    print(str2)
    return str1, str2


#end of def
#==========
#start of def

def convert_alignment_into_steps(str1, str2):
    pos=1
    step=1
    for i in range(len(str1)):
        if(step>50):
            break
        if(str1[i]==' '):
            print("\"Step "+ str(step)+ " : Insert "+ str(str2[i])+"\"")
            step+=1
            pos+=1
        elif(str2[i]==' '):
            print("\"Step "+ str(step)+ " : Delete "+ str(str1[i])+"\"")
            step+=1
        elif(str1[i]!=str2[i]):
            print("\"Step "+ str(step)+ " : Replace "+ str(str1[i])+" with "
   +str(str2[i])+"\"")
            step+=1
            pos+=1
```

```
        else:
            print("\"Step "+ str(step)+" : Match "+ str(str1[i])+"\"")
            step+=1
            pos+=1
        print()

#end of def
#==========
#start of def

def edit_distance_with_pointer(S,T):
    m=len(S)
    n=len(T)
    D=[[0 for j in range(n+1)]for i in range(m+1)]
    pointers=[['' for j in range(n+1)] for i in range(m+1)]
    D[0][0]= 0
    for i in range(m+1):
        D[i][0]=i
        pointers[i][0]='U'
    for j in range(1,n+1):
        D[0][j]=j
        pointers[0][j]='L'
    for i in range(1,m+1):
        for j in range(1,n+1):
            D[i][j]=min(min(D[i][j-1]+1,D[i-1][j]+1),s(S[i-1], T[j-1])+D[i
   -1][j-1])
            if(i==1 and j==1):
                pointers[i][j]='D'
            elif(D[i][j]==D[i-1][j]+1):
                pointers[i][j]='U'
            elif(D[i][j]==D[i][j-1]+1):
                pointers[i][j]='L'
            else:
                pointers[i][j]='D'
    print(D[m][n])
    print("=====")
    str1, str2 = convert_edit_sequence_to_alignment(S,T,pointers)
    print("=====")
    convert_alignment_into_steps(str1, str2)



#end of def
#==========
```

Listing 3: Program for Question 4

```
#==========
#start of def

def ParseProteinsTxt():
    '''
    This is a simple function to parse the Fasta files and return the amino
   acid sequence it reads.
    '''
    file_location='II-9-3-2022-proteins.txt'
    dict_of_sequences={}
    sequence_curr=""
    curr_sequence_name=""
    with open(file_location, "r") as fh:
        for line in fh:
            if(line.startswith("Protein")):
                curr_line=''.join(list(line))
                curr_sequence_name=curr_line[-2]
                dict_of_sequences[curr_sequence_name]=""
            elif(line.startswith("#")):
                if(sequence_curr!=""):
                    dict_of_sequences[curr_sequence_name]=sequence_curr
                sequence_curr=""
                curr_sequence_name=""
            else:
                seq_on_line = ''.join(list(line))
                seq_on_line=seq_on_line[:-1]
                sequence_curr+=seq_on_line
    return dict_of_sequences

#end of def
#==========
#start of def

def ParseBLOSUM():
    file_location='II-9-3-2022-blosum.txt'
    blosum_matrix={}
    with open(file_location, "r") as fh:
        count=0
        aa_list=""
        for line in fh:
                if(count==0):
                    seq_on_line = ''.join(list(line.split(' ')))
                    seq_on_line=seq_on_line[:-1]
                    aa_list=seq_on_line
                else:
                    seq_on_line = ''.join(list(line))
                    seq_on_line=seq_on_line[:-1]
                    if(len(seq_on_line)>0):
                        nums_temp_=line[1:-1].split(' ')
                        nums_temp=[]
                        for curr_rem in nums_temp_:
                            if(len(curr_rem)!=0):
                                nums_temp.append(curr_rem)
                        nums=[int(n) for n in nums_temp]
                        curr_AA_row=seq_on_line[0]
                        for i in range(len(aa_list)):
                            blosum_matrix[(curr_AA_row,aa_list[i])]=nums[i]
                count+=1
    return blosum_matrix
```

```
59
60 #end of def
61 #==========
62 #start of def
63
64 def new_s(Si,Tj):
65     return blosum_matrix[(Si,Tj)]
66
67 #end of def
68 #==========
69 #start of def
70
71 def max_score_with_scoring_matrix(S,T):
72     m=len(S)
73     n=len(T)
74     D=[['' for j in range(n+1)]for i in range(m+1)]
75     pointers=[['' for j in range(n+1)] for i in range(m+1)]
76     D[0][0]= 0
77     for i in range(m+1):
78         D[i][0]=i*-8
79         pointers[i][0]='U'
80     for j in range(1,n+1):
81         D[0][j]=j*-8
82         pointers[0][j]='L'
83     for i in range(1,m+1):
84         for j in range(1,n+1):
85             D[i][j]=max(max(D[i][j-1]-8,D[i-1][j]-8),new_s(S[i-1], T[j-1])+D
    [i-1][j-1])
86             if(i==1 and j==1):
87                 pointers[i][j]='D'
88             elif(D[i][j]==D[i-1][j]-8):
89                 pointers[i][j]='U'
90             elif(D[i][j]==D[i][j-1]-8):
91                 pointers[i][j]='L'
92             else:
93                 pointers[i][j]='D'
94     print(D[m][n])
95     print("=====")
96     str1, str2 = convert_edit_sequence_to_alignment(S,T,pointers)
97     print("=====")
98     convert_alignment_into_steps(str1, str2)
99
100 #Complexity of this is O(nm)
101
102
103 #end of def
104 #==========
```

Listing 4: Program for Question 5 and Question 6

```
#==========
#start of def

def scoring_for_gaps(S,T,u, flag_for_printing):
    '''
    V(i,j)=max(E(i,j), F(i,j), G(i,j))
    E(i,j)=max_{0<=k<j}(V(i,k) + w(j-k))
    F(i,j)=max_{0<=k<i}(V(k,j) + w(i-k))
    G(i,j)=V(i-1,j-1)+s(Si,Tj)

    normally: O(mn+ n+m)
    w(l)={0; l=0 \\ u; l>=1}
    u<0
    E(i,j)=max_{0<=k<=j-1}(V(i,k)+w(j-k))=max(E(i,j-1), V(i,j-1)+w(1)) (only
    insertions)
    F(i,j)=u+max_{0<=k<=i-1}(V(k,j)+w(i-k))=max(F(i-1,j), V(i-1,j)+w(1)) (
   only deletions)
    G(i,j)=V(i-1,j-1)+s(Si,Tj) (replacement on last character)
    V(i,j)=max(E(i,j), F(i,j), G(i,j))
    V(0,0)=0
    E(0,0)=0
    F(0,0)=0
    V(i,0)=u(i>0)
    V(0,j)=u(j>0)
    E(i,0)=u (i>0)
    F(0,j)=u(j>0)
    E(0,j)=0 (j>0)
    F(i,0)=0 (i>0)
    '''
    m=len(S)
    n=len(T)
    V=[[0 for j in range(n+1)] for i in range(m+1)]
    E=[[0 for j in range(n+1)]for i in range(m+1)]
    F=[[0 for j in range(n+1)]for i in range(m+1)]
    G=[[0 for j in range(n+1)]for i in range(m+1)]
    pointers=[['' for j in range(n+1)] for i in range(m+1)]
    for i in range(m+1):
        V[i][0]=u
        E[i][0]=2*u
        pointers[i][0]='U'
    for j in range(n+1):
        V[0][j]=u
        F[0][j]=2*u
        pointers[0][j]='L'
    V[0][0]=0
    E[0][0]=0
    F[0][0]=0
    for i in range(1,m+1):
        for j in range(1,n+1):
            E[i][j]=max(E[i][j-1], V[i][j-1]+u)
            F[i][j]=max(F[i-1][j], V[i-1][j]+u)
            G[i][j]=V[i-1][j-1]+new_s(S[i-1],T[j-1])
            V[i][j]=max(E[i][j], F[i][j], G[i][j])
            if(V[i][j]==G[i][j]):
                pointers[i][j]='D'
            elif(V[i][j]==E[i][j]):
                pointers[i][j]='L'
            else:
                pointers[i][j]='U'
```

```
    if(flag_for_printing):
        print(V[m][n])
        print("=====")
        str1, str2 = convert_edit_sequence_to_alignment(S,T,pointers)
        print("=====")
        convert_alignment_into_steps(str1, str2)
    return V[m][n]
#end of def
#==========
```

Listing 5: Program for Question 8

```
#==========
#start of def

def new_new_s(Si,Tj):
    if(Si==Tj):
        return 1
    else:
        return -1

#end of def
#==========
#start of def

def scoring_for_gaps_new(S,T,u, flag_for_printing):
    m=len(S)
    n=len(T)
    V=[[0 for j in range(n+1)] for i in range(m+1)]
    E=[[0 for j in range(n+1)]for i in range(m+1)]
    F=[[0 for j in range(n+1)]for i in range(m+1)]
    G=[[0 for j in range(n+1)]for i in range(m+1)]
    for i in range(m+1):
        V[i][0]=u
        E[i][0]=2*u
    for j in range(n+1):
        V[0][j]=u
        F[0][j]=2*u
    V[0][0]=0
    E[0][0]=0
    F[0][0]=0
    for i in range(1,m+1):
        for j in range(1,n+1):
            E[i][j]=max(E[i][j-1], V[i][j-1]+u)
            F[i][j]=max(F[i-1][j], V[i-1][j]+u)
            G[i][j]=V[i-1][j-1]+new_new_s(S[i-1],T[j-1])
            V[i][j]=max(E[i][j], F[i][j], G[i][j])
    return V[m][n]
#end of def
#==========
#start of def

def montecarloestimation(N, n, u, p_):
    Ui=['' for x in range(N)]
    Vi=['' for x in range(N)]
    V_gap=[0 for x in range(N)]
    rng = np.random.default_rng()
    for i in range(N):
        pred=rng.random()
        U=rng.choice(['a','b'],n, p=[p_,1-p_])
        Ui[i]=''.join(U)
        V=rng.choice(['a','b'],n, p=[p_,1-p_])
        Vi[i]=''.join(V)
        V_gap[i]=scoring_for_gaps_new(Ui[i], Vi[i], u, False)
    mean=sum(V_gap)/N
    return (mean/n)

#end of def
#==========
```

Listing 6: Program for Question 11

```
#==========
#start of def

def scoring_for_alignment(S,T,u_del, u_ins):
    '''
    V_s(i,j)=V_suf(i,j)=max(0,V_s(i-1,j)+s(Si,-), Vs(i,j-1)+s(-,Tj), Vs(i-1,j-1)+s(Si,Tj))
    s(Si,-)=u_del
    s(-,Tj)=u_ins
    s being used is new_s from Section 2 of the project
    v_sub(S,T)=max_i,j {V_s(i,j)}
    Assume u_ins and u_del are constants (ie s(Si,-) and s(-,Tj) are constants for all Si,Tj)
    '''
    m=len(S)
    n=len(T)
    V_s=[[0 for j in range(n+1)] for i in range(m+1)]
    vsub=0
    for i in range(m+1):
        V_s[i][0]=0
    for j in range(n+1):
        V_s[0][j]=0
    V_s[0][0]=0
    for i in range(1,m+1):
        for j in range(1,n+1):
            V_s[i][j]=max(0, V_s[i][j-1]+u_ins, V_s[i-1][j]+u_del, V_s[i-1][j-1]+new_s(S[i-1],T[j-1]))
            vsub=max(vsub,V_s[i][j])
    return vsub

#end of def
#==========
```