9 Operational Research

9.3 Protein Comparison in Bioinformatics

Question 1:

$Claim\ 1$:
$s(a,b) = 1\ if\ a \neq b$;
$s(a,b) = 0\ otherwise$.

Proof:

To get to $D(i,j)$, we must consider four possibility:

1) $transforming\ S[1,i-1]\ to\ T[1,j]\ and\ delete\ S(i)$
2) $transforming\ S[1,i]\ to\ T[1,j-1]\ and\ insert\ T(j)$
3*) $if\ S(i) = T(j), D(i,j) = D(i-1,j-1)\ as\ matching\ does\ not\ change\ edit\ distance$
4*) $if\ S(i) \neq T(j), D(i,j) = D(i-1,j-1) + 1\ as\ we\ can\ replace\ S(i)\ by\ T(j)$

For getting $D(i,j)$, we must get the minimum edit distance among the three cases.
(We only need to consider one of the two starred cases each time as it is depending on whether $S(i) = T(j)$ or not.)
Case 1 has edit distance $\boxed{D(i-1,j)+1}$
Case 2 has edit distance $\boxed{D(i,j-1)+1}$
Case 3 (/4) has edit distance $\boxed{D(i-1,j)+s(S_i,T_j)}$

As replacement takes up 1 edit distance and matching does not take up any edit distance,
$s(S_i,T_j) = 1\ if\ S_i \neq T_j$
$s(S_i,T_j) = 0\ if\ S_i = T_j$

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

■

Boundary conditions are $D(0,0) = 0; D(i,0) = i; D(0,j) = j$. This is because it takes i deletions to get from $S[1,i]$ to nothing and similarly it takes j insertions to get from nothing to $T[1,j]$.

$Remark$: $we\ can\ summarise\ into\ 3\ cases\ because\ each\ D(i,j)\ must\ followed\ from\ the\ optimal\ path\ which\ one\ of\ its\ neighbours\ has\ followed\ from\ D(0,0)$.

Question 2: Run the program [editdistanceQ2.m]

The edit distance between 'shesells' and 'seashells' is 3.

The complexity of the program editdistanceQ2.m is $O(mn)$, where m and n are the length of $S\ and\ T$ respectively.

This is because for finding the edit distance, we need to calculate the matrix D by using the algorithm suggested in Question 1, $D(i,j) = \min(D(i-1,j)+1, D(i,j-1)+1, D(i-1,j-1)+s(S_i,T_j))$.

Assigning boundary conditions: m+n+1 operations
3 additions, 1 assignment (for $s(S_i,T_j)$) and 1 comparison for an entry: 5(m-1)x(n-1)

Therefore, the total complexity $= m+n+1+5(m-1)*(n-1) = 5mn-4m-4n+1 = O(mn)$

Questions 3: Run the program [editdistance_modQ3.m]

To modify my algorithm so as to produce one possible optimal alignment between two strings, I need to find out the path we get from $D(0,0)$ to $D(m,n)$. For convenience, I have designed another matrix R to store the values in D such that $D(i,j) = R(i+1, j+1)$. Therefore, my new algorithm aims to find a path from $R(1,1)$ to $R(m+1, n+1)$.

To find the path, we need to work backwards and track how we get to the bottom right of matrix R. From the values in $R(x,y), R(x-1,y-1), R(x,y-1) and\ R(x-1,y)$, we can easily deduce what the operation has been taken to reach $R(x,y)$.

If

1) $R(x-1,y-1) = R(x,y) - s(S_x, T_y)$
   We can deduce either replacement or matching has been taking place depending on $S_{x-1}\ and\ T_{y-1}$ which takes us from $R(x-1,y-1)$ to $R(x,y)$.
2) $R(x-1,y) = R(x,y) - 1$
   We can deduce that deletion has been taking place which takes us from $R(x-1,y)$ to $R(x,y)$.
3) Similarly, if $R(x,y-1) = R(x,y) - 1$
   We can deduce that insertion has been taking place which takes us from $R(x,y-1)$ to $R(x,y)$.

It is possible that two or more of the three cases happened at the same time. (Example: 'ab' and 'ba') In fact, no matter which path we take among the three, we should get the same edit distance as each step is only depending on the previous step's edit distance only but not those before the previous step. As a rule of thumb, my algorithm follows the following priority: replacement/Matching, Deletion and Insertion.

Working backwards to get all the operations in reverse order with this algorithm and present the result in the correct order at the end will gives a possible optimal alignment between two stings.

After running the modified algorithm, the minimum edit distance between protein A and protein B is 84.

First 50 steps of an optimal alignment:

	With Matching		*Without Matching*
1	Match M with M	1	Delete M
2	Delete G	2	Delete G
3	Delete L	3	Replace L with M
4	Replace S with A	4	Delete S
5	Match D with D	5	Delete D
6	Delete G	6	Replace G with A
7	Delete E	7	Replace E with D
8	Replace W with F	8	Replace W with F
9	Replace Q with D	9	Replace Q with D
10	Replace L with A	10	Replace L with A
11	Match V with V	11	Replace V with V
12	Match L with L	12	Replace L with L
13	Replace N with K	13	Replace N with K

14	Replace V with C	14	Replace V with C
15	Match W with W	15	Replace W with W
16	Match G with G	16	Replace G with G
17	Replace K with P	17	Replace K with P
18	Match V with V	18	Replace V with V
19	Match E with E	19	Replace E with E
20	Replace T with A	20	Replace T with A
21	Match D with D	21	Replace D with D
22	Replace L with Y	22	Replace L with Y
23	Replace A with T	23	Replace A with T
24	Replace G with T	24	Replace G with T
25	Replace H with M	25	Replace H with M
26	Match G with G	26	Replace G with G
27	Replace Q with G	27	Replace Q with G
28	Replace E with L	28	Replace E with L
29	Match V with V	29	Replace V with V
30	Match L with L	30	Replace L with L
31	Replace I with T	31	Replace I with T
32	Replace S with R	32	Delete S
33	Match L with L	33	Replace L with R
34	Match F with F	34	Replace F with L
35	Match K with K	35	Replace K with F
36	Replace G with E	36	Replace G with K
37	Match H with H	37	Replace H with E
38	Match P with P	38	Replace P with H
39	Match E with E	39	Replace E with P
40	Match T with T	40	Replace T with E
41	Replace L with Q	41	Replace L with T
42	Replace E with K	42	Replace E with Q
43	Replace K with L	43	Replace K with K
44	Match F with F	44	Replace F with L
45	Replace E with P	45	Replace E with F
46	Match K with K	46	Replace K with P
47	Match F with F	47	Replace F with K
48	Replace K with A	48	Replace K with F
49	Replace H with G	49	Replace H with A
50	Replace L with I	50	Replace L with G
:	:	:	:
:	:	:	:
151	Match G with G	81	Replace G with A
152	Match F with F	82	Replace Q with K
153	Replace Q with S	83	Replace H with G
154	Match G with G	84	Replace E with S

Table 1: Optimal Alignment from protein A to protein B

Question 4: Run the program [scoreQ4.m]

Noticing we are now maximizing the score for our optimal alignment instead of minimizing the edit distance, the result is likely to be different from the previous question.

Modifying the program [editdistance_modQ3.m] as we have changed our algorithm slightly to
$D(i,j) = \max\left(D(i-1,j) - 8, D(i,j-1) + 1, D(i-1,j-1) + s(S_i, T_j)\right)$
$where\ s(S_i, T_j) = BLOSUM\ matrix$.

Similar to Question 3, to find the path, we need to work backwards and track how we get to the bottom right of the matrix R.

If

1) $R(x-1, y-1) = R(x,y) - s(S_x, T_y)$
   We can deduce either replacement or matching has been taking place depending on $S_{x-1}\ and\ T_{y-1}$ which takes us from $R(x-1, y-1)$ to $R(x,y)$.
2) $R(x-1, y) = R(x,y) + 8$
   We can deduce that deletion has been taking place which takes us from $R(x-1, y)$ to $R(x,y)$.
3) Similarly, if $R(x, y-1) = R(x,y) + 8$
   We can deduce that insertion has been taking place which takes us from $R(x, y-1)$ to $R(x,y)$.

Similarly, if more than one case appear to be true, we can take any of those operations as the optimum alignment does not depend on the steps before the previous steps.

With the similar approach as Question 3, we present one of the optimal alignments for the maximum score.

After running [scoreQ4.m], the maximum score between protein A and protein B is 287.
First 50 steps of an optimal alignment:

1	Match M with M
2	Delete G
3	Delete L
4	Replace S with A
5	Match D with D
6	Delete G
7	Delete E
8	Replace W with F
9	Replace Q with D
10	Replace L with A
11	Match V with V
12	Match L with L
13	Replace N with K
14	Replace V with C
15	Match W with W
16	Match G with G

17	Replace K with P
18	Match V with V
19	Match E with E
20	Replace T with A
21	Match D with D
22	Replace L with Y
23	Replace A with T
24	Replace G with T
25	Replace H with M
26	Match G with G
27	Replace Q with G
28	Replace E with L
29	Match V with V
30	Match L with L
31	Replace I with T
32	Replace S with R
33	Match L with L
34	Match F with F
35	Match K with K
36	Replace G with E
37	Match H with H
38	Match P with P
39	Match E with E
40	Match T with T
41	Replace L with Q
42	Replace E with K
43	Replace K with L
44	Match F with F
45	Replace E with P
46	Match K with K
47	Match F with F
48	Replace K with A
49	Replace H with G
50	Replace L with I
:	:
:	:
:	:
149	Match E with E
150	Match L with L
151	Match G with G
152	Match F with F
153	Replace Q with S
154	Match G with G

Table 2: Optimal alignment from protein A to protein B with maximum score

Question 5:

As we are taking gaps into account as well, we need to consider a new algorithm. In the modified algorithm, the score and the options we take for each entry in matrix R no longer be only depending on the previous step, but also some steps before the previous step.

New algorithm:

$V_{gap}(i,j) = \max\{E(i,j), F(i,j), G(i,j)\}$
$E(i,j) = \max\limits_{0\le k\le j-1}\{V_{gap}(i,k) + w(j-k)\}$ $F(i,j) = \max\limits_{0\le k\le i-1}\{V_{gap}(k,j) + w(i-k)\}$ $G(i,j) = V_{gap}(i-1, j-1) + s(S_i, T_j)$

For each value in $V_{gap}(i,j)$ (which is same as each entry in matrix $R(i+1, j+1)$), we would need to take up to $j-1$ calculations and one comparison operation for $E(i,j)$; and similarly $i-1$ calculations and one comparison operation for $F(i,j)$. Finally, it takes 1 operation to get $G(i,j)$ and 1 comparison operation for getting $V_{gap}(i,j)$. As there are mn values in $V(i,j)$ grid, it has complexity of $mn(n+m+2)$ which is $O(mn^2 + nm^2)$.

If we let $w(l)$ takes some fixed value u for all $l \geq 1$, we can simplified our algorithm into the following:

$V_{gap}(i,j) = \max\{E(i,j), F(i,j), G(i,j)\}$	
$E(i,j) = \max\limits_{0\le k\le j-1}\{V_{gap}(i,k) + u\}$	Matching the gap
$F(i,j) = \max\limits_{0\le k\le i-1}\{V_{gap}(k,j) + u\}$	Matching the gap
$G(i,j) = V_{gap}(i-1, j-1) + s(S_i, T_j)$	Replace/Match

In other word, we may consider our algorithm as follow:

$V_{gap}(i,j) = \max\{E'(i,j), F'(i,j), G'(i,j)\}$	
$E'(i,j) = \max\limits_{0\le k\le j-1}\{V_{gap}(i,k)\}$	Matching the gap
$F'(i,j) = \max\limits_{0\le k\le i-1}\{V_{gap}(k,j)\}$	Matching the gap
$G'(i,j) = V_{gap}(i-1, j-1) + s(S_i, T_j) - u$	Replace/Match

Where

$E'(i,j)$ implies an insertion after the optimal alignment changing $S[1,i]\ to\ T[1,k]$ (inserting $T[k+1,j]$ at the end);
$F'(i,j)$ implies a deletion after the optimal alignment changing $S[1,k]\ to\ T[1,j]$ (deleting $S[k+1,i]$ at the end);
$G'(i,j)$ implies a replacement or a matching operation taken for getting $S[1,i-1]\ to\ T[1,j-1]$.

Therefore, we now have only 1 comparison operation for calculating E' and F' and G' requires 1 addition operation for the algorithm for each entry. In addition, as there are mn entries in total, the complexity of the new algorithm is $O(mn)$.

Boundary conditions are $V_{gap}(0,0) = 0; V_{gap}(i,0) = V_{gap}(0,j) = u$ for all $i \geq 1; j \geq 1$.

Question 6: Run the program [gapscoreQ5.m]

After running [gapscoreQ5.m], the maximum gap-weighted score between protein C and protein D is 622.
First 50 steps of an optimal alignment:

1	Match M with M
2	Replace T with S
3	Replace T with R
4	Replace C with Q
5	Match S with S
6	Insert S
7	Insert V
8	Insert S
9	Insert F
10	Insert R
11	Insert S
12	Insert G
13	Insert G
14	Insert S
15	Match R with R
16	Replace Q with S
17	Match F with F
18	Replace T with S
19	Replace S with T
20	Replace S with A
21	Match S with S
22	Insert A
23	Insert I
24	Insert T
25	Insert P
26	Insert S
27	Insert V
28	Insert S
29	Insert R
30	Insert T
31	Insert S
32	Insert F
33	Insert T
34	Insert S

35	Insert V
36	Insert S
37	Insert R
38	Insert S
39	Insert G
40	Insert G
41	Insert G
42	Insert G
43	Insert G
44	Insert G
45	Insert G
46	Insert F
47	Insert G
48	Insert R
49	Insert V
50	Match S with S
:	:
:	:
:	:
631	Delete K
632	Delete V
633	Delete V
634	Delete S
635	Delete T
636	Delete H
637	Delete E
638	Delete Q
639	Delete V
640	Delete L
641	Replace R with S
642	Replace T with F
643	Match K with K
644	Replace N with S

Table 3: optimal alignment from protein C to protein D with the maximum gap-weighted score

Question 7:

Claim 2:
for all $0 \le p \le 1$, *and for all* $u \le 0$,
$\liminf_{n\to\infty} \frac{\mathbb{E}\left(v_{gap}(U^n, V^n)\right)}{n} > 0$

Noticing for any 'replacing' operation, we can replace it by a 'deletion' and a 'insertion' operation. Therefore, the score of replacement of a sequence of length l of the same protein in this example will be $R_l = \max(-l, 2u)$.

Proof:

Lemma $2a$:
for all $0 \le p \le 1$, *and for all* $u \le 0$,
$\liminf_{n\to\infty} \mathbb{E}(v_{gap}(U^n, V^n)) > \liminf_{n\to\infty} \mathbb{E}(v(U^n, V^n))$

Proof:

Instead of introducing the fixed score of inserting/deleting a sequence of length l be u, I let u be the score of inserting/deleting a protein. For any protein transformation which involved deletion or insertion of a sequence of length greater than 1, they will have a score $v(U^n, V^n) \le v_{gap}(U^n, V^n)$. Equality occurs if $R_l = \max(-l, 2u) = -l$ when inserting/deleting a sequence protein does not give a higher score than simply do l replacement operations. However, as $n \to \infty$, considering all possible protein transformation of length n, there must be a transformation involving replacement of $q+1$ such that $R_{q+1} = \max(-(q+1), 2u)$=2u where q > 2u + 1. Therefore, for $n \to \infty$, there must exist some (U^n, V^n), such that $v(U^n, V^n) < v_{gap}(U^n, V^n)$.

Hence,

$$\liminf_{n\to\infty} \mathbb{E}(v_{gap}(U^n, V^n)) = \liminf_{n\to\infty} \sum_{all\ combinations} P(u = U^n, v = V^n)\, v_{gap}(U^n, V^n))$$

$$> \liminf_{n\to\infty} \sum_{all\ combinations} P(u = U^n, v = V^n)\, v(U^n, V^n))$$

$$= \liminf_{n\to\infty} \mathbb{E}(v(U^n, V^n))$$

■

Lemma 2b:
for all $0 \le p \le 1$, *and for all* $u \le 0$,
$\liminf_{n\to\infty} \mathbb{E}(v(U^n, V^n)) \ge 0$

Proof:

By induction:
For n=1,

U^1	V^1	v	$Probability\ of\ (U^1, V^1)$
a	a	1	p^2
a	b	R_1	$p(1-p)$
b	a	R_1	$p(1-p)$
b	b	1	$(1-p)^2$

Taking all possible sequence pairs and their transformation scores, we get an inequality

$$\mathbb{E}\big(v(U^1, V^1)\big) = p^2 + (1-p)^2 + 2R_1 p(1-p)$$
$$= \begin{cases} (2p-1)^2 \ge 0, & u < -0.5 \\ p^2 + (1-p)^2 + 4up(1-p) \ge 0, & u \ge -0.5 \end{cases}$$

(Remark: $4up(1-p) \ge -2p(1-p)\ as\ 2u \ge -1.$)
Therefore, $\mathbb{E}\big(v(U^1, V^1)\big) \ge 0$.

If $\mathbb{E}\left(v\big(U^k, V^k\big)\right) \ge 0$ is true, consider $\mathbb{E}\left(v\big(U^{k+1}, V^{k+1}\big)\right)$

$U^k + U^1$	$V^k + V^1$	v	$Probability\ of\ (U^{k+1}, V^{k+1})$
U^ka	V^ka	v' +1	$P(U^k, V^k)p^2$
U^ka	V^kb	$v' + R_1$	$P(U^k, V^k)p(1-p)$
U^kb	V^ka	$v' + R_1$	$P(U^k, V^k)p(1-p)$
U^kb	V^kb	v' +1	$P(U^k, V^k)(1-p)^2$

denoting $v' = v\big(U^k, V^k\big)$

$$\mathbb{E}\left(v\big(U^{k+1}, V^{k+1}\big)\right) = \sum P\big(U^{k+1}, V^{k+1}\big)\, v\big(U^{k+1}, V^{k+1}\big)$$
$$= \sum P\big(U^k, V^k\big)\{(v'+1)p^2 + 2(v'+R_1)p(1-p) + (v'+1)(1-p^2)\}$$
$$= \sum P\big(U^k, V^k\big)\{v'(p+1-p)^2 + p^2 + (1-p)^2 + 2R_1 p(1-p)\}$$
$$= \sum P\big(U^k, V^k\big)\big\{v' + \mathbb{E}\big(v(U^1, V^1)\big)\big\}$$
$$\ge \sum P\big(U^k, V^k\big)\, v' = \mathbb{E}\left(v\big(U^k, V^k\big)\right) \ge 0$$

Therefore, $\liminf_{n\to\infty} \mathbb{E}(v(U^n, V^n)) \ge 0$

■

Hence, combining the two lemmas

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

■

Question 8:

For estimating $n^{-1}\mathbb{E}(v_{gap}(U^n, V^n))$, I have written a program for averaging a given number of samples (k) of $\mathbb{E}(v_{gap}(U^n, V^n))$ and divide it by n at the end. To avoid unreasonable computation time, I have chosen the number of samples $k = \left\lceil \frac{10^6}{n^2} \right\rceil$ as program [Q8.m] is working at order $o(kn^2)$. As it is impossible and not reliable if we only take 1 result for a large n without averaging, I decided to carry out the approximation at small n and obtain an extrapolation for n tends to infinity from the known results.

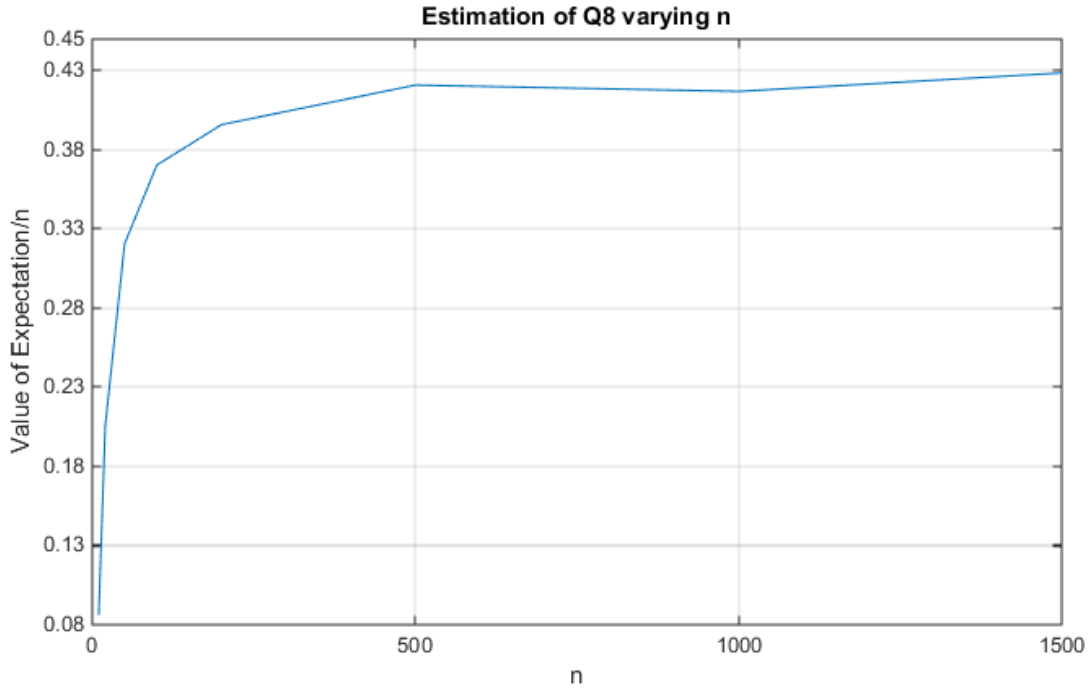

Figure 1: Estimation of the function in Q8 with different n (graph)

N	K	$n^{-1}\mathbb{E}(v_{gap}(U^n, V^n))$
10	10000	0.086620
20	2500	0.205300
50	400	0.320700
100	100	0.370500
200	25	0.396000
500	4	0.421000
1000	1	0.417000
1500	1	0.428667

Table 4: Estimation of the function in Q8 with different n (chart)

From the table, we can deduce that the expected value does converge as N increases. By plotting the result and carefully extrapolate the line, I believe the expected value to be around 0.430 when n tends to infinity.

Question 9:

$Claim\ 3$:
$v_{sub}(S,T) = \max\{v_{sfx}(S',T'): S'a\ prefix\ of\ S, T'a\ prefix\ of\ T\}$

$where$
$v_{sub}(S,T) = \max\{v(S',T'): S'a\ substring\ of\ S, T'a\ substring\ of\ T\}$;
$v_{sfx}(S,T) = \max\{v(S',T'): S'a\ suffix\ of\ S, T'a\ suffix\ of\ T\}$.

Proof:

$Lemma\ 3a$:
$for\ any\ substring\ S' \epsilon\ S, S'can\ be\ extended\ to\ the\ first\ character\ and\ become\ the\ prefix\ of\ S\ (denoted\ S'').-(1)$
$And\ original\ S'becomes\ the\ suffix\ of\ the\ new\ substring\ S''.-(2)$

Proof:

(1): Consider a substring $S' \epsilon\ S$ and we can try to build a new substring $\bar{S}$ in the following manner:

Include $S(1)$ into $\bar{S}$ as the first element;

-if $S' \epsilon\ \bar{S}$, we stop

-if $S' is\ not\ included\ in\ \bar{S}$, we continue to add the next element from S to $\bar{S}$.

Existence & (2): From the above algorithm, as the substring $S' \epsilon\ S$, it must stop at some point in the string with the suffix of $\bar{S}$ be the substring S'.

(1): In addition, as we start building substring $\bar{S}$ from the first element of S, $\bar{S}$ is also the prefix of S.

■

By definition, $v_{sub}(S,T) = \max\{v(S',T'): S'a\ substring\ of\ S, T'a\ substring\ of\ T\}$. As we proved in $Lemma\ 3a$, any substring can be extended to the suffix of a prefix of S. Therefore, we can rewrite

$v_{sub}(S,T) = \max\{v(S'',T''): S''a\ suffix\ of\ a\ prefix\ S', T''\ a\ suffix\ of\ a\ prefix\ T' of\ S\}$
$\rightarrow v_{sub}(S,T) = \max\{v_{sfx}(S',T'): S'a\ prefix\ of\ S, T'a\ prefix\ of\ T\}$

■

Question 10:

$Claim\ 4$:

$$V_{sfx}(i,j) = \max \begin{cases} V_{sfx}(i-1,j) + s(S_i,-) & Case\ 1 \\ V_{sfx}(i,j-1) + s(-,T_j) & Case\ 2 \\ V_{sfx}(i-1,j-1) + s(S_i,T_j) & Case\ 3 \\ 0 & Case\ 4 \end{cases}$$

With Boundary conditions $V_{sfx}(i,0) = V_{sfx}(0,j) = 0$.

Proof:

Notice that we do not take gap into account in this Question, we are having a similar problem as Question 1. This time, we only concern about the positive score because the minimum score for $V_{sfx}(i,j)$ is always 0 as we can always consider take substring of length 0 from S and from T which gives a score of 0.

To get $V_{sfx}(i,j)$, we must consider four possibility:

1) $transforming\ S[k_1, i-1]\ and\ T[t_1, j]\ and\ delete\ S(i)$
2) $transforming\ S[k_2, i]\ and\ T[t_2, j-1]\ and\ insert\ T(j)$
3) $transforming\ S[k_3, i-1]\ and\ T[t_3, j-1]\ and\ matching\ or\ replace\ S(i)\ with\ T(j)\ D(i,j)$
4) $consider\ substrings\ of\ length\ 0\ from\ S\ and\ T\ and\ gives\ a\ score\ of\ 0$

$Assuming\ V_{sfx}(i-1,j) = v(S[k_1, i-1], T[t_1, j])$;
$V_{sfx}(i,j-1) = v(S[k_2, i], T[t_2, j-1])$
$V_{sfx}(i-1,j-1) = v(S[k_3, i], T[t_3, j-1])$

Why only 4 possibility?
This is because if $v(S'[x_1, i], T'[x_2, j])$ is the maximum score obtained in $V_{sfx}(i,j)$, for $V_{sfx}(i+1, j+1)$, one of the possible value will be $v(S'[x_1, i], T'[x_2, j]) + s(S_{i+1}, T_{j+1})$, which is described in case 3.
If $V_{sfx}(i+1, j+1) = v(S'[y_1, i], T'[y_2, j]) + s(S_{i+1}, T_{j+1})$, (where $x_1 \neq y_1, x_2 \neq y_2$), $V_{sfx}(i,j)$ will not be equal to $v(S'[x_1, i], T'[x_2, j])$ as $v(S'[y_1, i], T'[y_2, j]) \geq v(S'[x_1, i], T'[x_2, j])$.

Similar argument for case (1) and (2).

For case (4), if the score is negative for all case (1) to (3), given that we can take substring of length 0, we can always pick a score of 0 for $V_{sfx}(i,j)$.

Therefore, we only need to compare these 4 cases.

■

Boundary condition:

$V_{sfx}(i,0) = V_{sfx}(0,j) = 0, for\ all\ i,j$

This is because $V_{sfx}(i,0) = \max_k\{v(S[k,i],-)\}$ and $V_{sfx}(0,j) = \max_k\{v(-,T[k,j])\}$ and by observation, given $s(S_i,-) and\ s(-,T_j) < 0$, the values of them will be 0.

Question 11: Run the program [scoreQ11.m]

I have created a matrix $D\ such\ that\ D(i+1, j+1) = V_{sfx}(i, j)$ and run the algorithm suggested in Question 10. The maximum score will be the maximum entry in $matrix\ D$. Noticing the complexity of [scoreQ11.m] is $O(mn)$.

After running the program, $v_{sub} = 799$ for proteins C and D.

Program:

1.

```
function [D]=editdistanceQ2
A=input('please enter the first string: ');
B=input('please enter the second string: ');
M=length(A);
N=length(B);
D=zeros(M+1,N+1);
%boundary condition
D(1,1)=0;
for t=2:M+1
    D(t,1)=t-1;
end
for t=2:N+1,
    D(1,t)=t-1;
end
for k=2:M+1;
    for s=2:N+1,
        %set the three comparison
        W=D(k-1,s)+1;
        X=D(k,s-1)+1;
        if A(k-1)==B(s-1),
            Y=D(k-1,s-1);
        else
            Y=D(k-1,s-1)+1;
        end
        D(k,s)=min([W,X,Y]);
    end
end
FF=['The minimum edit distance from ',A,' to ',B,' is ',
num2str(D(M+1,N+1))];
disp(FF)
end
```

2.

```
function [D]=editdistance_modQ3
A=input('please enter the first string: ');
B=input('please enter the second string: ');
M=length(A);
N=length(B);
D=zeros(M+1,N+1);
%boundary condition
D(1,1)=0;
for t=2:M+1
    D(t,1)=t-1;
end
for t=2:N+1,
    D(1,t)=t-1;
end
for k=2:M+1;
    for s=2:N+1,
        %set the three comparison
        W=D(k-1,s)+1;
        X=D(k,s-1)+1;
        if A(k-1)==B(s-1),
            Y=D(k-1,s-1);
        else
            Y=D(k-1,s-1)+1;
        end
        D(k,s)=min([W,X,Y]);
    end
end
FF=['The minimum edit distance from A to B is ', num2str(D(M+1,N+1))];
disp(FF)

a=M+1;
b=N+1;
K=[0];
while (a~=1)&&(b~=1),
    P=D(a-1,b-1);
    O=D(a-1,b);
    I=D(a,b-1);
    if A(a-1)==B(b-1)
        s=0;
    else
        s=1;
    end
    if P==D(a,b)-s,
        if P==D(a,b),
            %match:1
            a=a-1;
            b=b-1;
            K=[1;K];
        else %replace:2
            a=a-1;
            b=b-1;
            K=[2;K];
        end
    else if (O==D(a,b)-1)
            %deletion:3
```

```
            a=a-1;
            b=b;
            K=[3;K];
        else %insert:4
        a=a;
        b=b-1;
        K=[4;K];
        end
    end
end

TT=size(K);
i=1;
j=1;
for n=1:TT(1,1),
    switch(K(n,1))
        case 1
            disp(['Match ', A(i), ' with ', B(j)'])
            i=i+1;
            j=j+1;
        case 2
            disp(['Replace ', A(i), ' with ', B(j)'])
            i=i+1;
            j=j+1;
        case 3
            disp(['Delete ', A(i)])
            i=i+1;
            j=j;
        case 4
            disp(['Insert ', B(j)])
            j=j+1;
            i=i;
    end
end
```

3.

```
function [D]=scoreQ4
A=input('please enter the first string: ');
B=input('please enter the second string: ');
M=length(A);
N=length(B);
D=zeros(M+1,N+1);
Order='CSTPAGNDEQHRKMILVFYW';
B62=blosum(62,'order',Order);


%boundary condition
D(1,1)=0;
for t=2:M+1
    D(t,1)=(t-1)*(-8);
end
for t=2:N+1,
    D(1,t)=(t-1)*(-8);
end
for k=2:M+1;
    for s=2:N+1,
        %set the three comparison
        W=D(k-1,s)-8;
        X=D(k,s-1)-8;
        Y_1= strfind(Order,A(k-1));
        Y_2= strfind(Order,B(s-1));
        Y=D(k-1,s-1)+B62(Y_1,Y_2);
        D(k,s)=max([W,X,Y]);
    end
end
FF=['The maximum score from A to B is ', num2str(D(M+1,N+1))];
disp(FF)

a=M+1;
b=N+1;
K=[0];
R=0;
while (a~=1)&&(b~=1),
    P=D(a-1,b-1);
    O=D(a-1,b);
    I=D(a,b-1);
    R=0;
    if O==(D(a,b)+8),
        %deletion:3
            a=a-1;
            K=[3;K];
            R=1;
    else if I==(D(a,b)+8),
        %insert:4
        b=b-1;
        K=[4;K];
        R=1;
        end
    end
    if R==0,
```

```
        if A(a-1)==B(b-1),
            %matching
            a=a-1;
            b=b-1;
            K=[1;K];
        else
            %replace
            a=a-1;
            b=b-1;
            K=[2;K];
        end
        R=1;
    end
end

TT=size(K);
i=1;
j=1;
for n=1:TT(1,1),
    switch(K(n,1))
        case 1
            disp(['Match ', A(i), ' with ', B(j)'])
            i=i+1;
            j=j+1;
        case 2
            disp(['Replace ', A(i), ' with ', B(j)'])
            i=i+1;
            j=j+1;
        case 3
            disp(['Delete ', A(i)])
            i=i+1;
            j=j;
        case 4
            disp(['Insert ', B(j)])
            j=j+1;
            i=i;
    end
end
```

4.

```
function []=gapscoreQ5
A=input('please enter the first string: ');
B=input('please enter the second string: ');
C=input('please enter the value of u: ');
M=length(A);
N=length(B);
D=zeros(M+1,N+1);
Order='CSTPAGNDEQHRKMILVFYW';
B62=blosum(62,'order',Order);

X=zeros(M+1,N+1);
Y=zeros(M+1,N+1);

%boundary condition
D(1,1)=0;
%define matrix X as action taken for getting to certain entry
%define matrix Y as the gap distance inserting/deleting
for t=2:M+1
    D(t,1)=(C);
    X(t,1)=3;
    Y(t,1)=1;
end
for t=2:N+1,
    D(1,t)=(C);
    X(1,t)=4;
    Y(1,t)=1;

end

for k=2:M+1;
    for s=2:N+1,
        %set the three comparison
        F=max(D(1:k-1,s))+C;
        F_pos=find(D(1:k-1,s)+C==F);
        E=max(D(k,1:s-1))+C;
        E_pos=find(D(k,1:s-1)+C==E);
        G_1= strfind(Order,A(k-1));
        G_2= strfind(Order,B(s-1));
        G=D(k-1,s-1)+B62(G_1,G_2);
        D(k,s)=max([E,F,G]);
        switch (D(k,s))
            case E
            %insertion
            X(k,s)=4;
            Y(k,s)=E_pos(1,1);
            case F
                %deletion
                X(k,s)=3;
                Y(k,s)=F_pos(1,1);
            case G
               if A(k-1)==B(s-1),
                  %match
                  X(k,s)=1;
```

```
                else %replace
                   X(k,s)=2;
                end
            end
        end
end
FF=['The maximum gap-weighted score from A to B is ', num2str(D(M+1,N+1))];
disp(FF)
a=M+1;
b=N+1;
K=[0,0];
JJ=1;
while JJ==1,
    switch(X(a,b))
        case 1
            K=[a,b;K];
            a=a-1;
            b=b-1;
        case 2
            K=[a,b;K];
            a=a-1;
            b=b-1;
        case 3
            K=[a,b;K];
            a=Y(a,b);
            b=b;
        case 4
            K=[a,b;K];
            a=a;
            b=Y(a,b);
    end
    if and(a==1,b==1)
        JJ=0;
        K=[1,1;K];
    end
end
TT=size(K);
i=1;
j=1;
for n=1:TT(1,1)-1,
    switch(X(K(n,1),K(n,2)))
        case 1
            disp(['Match ', A(i), ' with ', B(j)'])
            i=i+1;
            j=j+1;
        case 2
            disp(['Replace ', A(i), ' with ', B(j)'])
            i=i+1;
            j=j+1;
        case 3
            if n==1,
                U=K(n,1);
            else
            U=K(n,1)-K(n-1,1);
            end
            for u=1:U,
                disp(['Delete ', A(i+u-1)])
```

```
            end
            i=i+U;
            j=j;
        case 4
            if n==1,
                U=K(n,2);
            else
            U=K(n,2)-K(n-1,2);
            end

            for u=1:U
                disp(['Insert ', B(j+u-1)])
            end
            j=j+U;
            i=i;
    end
end
FF=['The maximum gap-weighted score from A to B is ', num2str(D(M+1,N+1))];
disp(FF)
end
```

5.

```
function [Res]=Q8(n,u)
K=0;
tic
N=ceil(1000000/(n^2))
for L=1:N
    A=randi([0 1],n,1);
    B=randi([0 1],n,1);
    K=gapscoreQ8_inner(A,B,u)+K;
end
K=K/N;
Res=(1/(n))*K;
toc
end
function [ANS]=gapscoreQ8_inner(A,B,C)
M=length(A);
N=length(B);
D=zeros(M+1,N+1);
%boundary condition
D(1,1)=0;
for t=2:M+1
    D(t,1)=C;
end
for t=2:N+1,
    D(1,t)=C;
end
for k=2:M+1;
    for s=2:N+1,
        %set the three comparison
        F=max(D(1:k-1,s))+C;
        E=max(D(k,1:s-1))+C;
        if A(k-1)==B(s-1),
            G_1=1;
        else G_1=-1;
        end
        G=D(k-1,s-1)+G_1;
        D(k,s)=max([E,F,G]);
    end
end
ANS=D(M+1,N+1);
end
```

6.

```
function scoreQ11
A=input('please enter the first string: ');
B=input('please enter the second string: ');
M=length(A);
N=length(B);
D=zeros(M+1,N+1);
Order='CSTPAGNDEQHRKMILVFYW';
B62=blosum(62,'order',Order);


for k=2:M+1;
    for s=2:N+1,
        %set the three comparison
        W=D(k-1,s)-2;
        X=D(k,s-1)-2;
        Y_1= strfind(Order,A(k-1));
        Y_2= strfind(Order,B(s-1));
        Y=D(k-1,s-1)+B62(Y_1,Y_2);
        D(k,s)=max([W,X,Y,0]);
    end
end
FF=['The v_sub from A to B is ', num2str(max(D(:)))];
disp(FF)

end
```