BCB410

Exercise questions for **"Dynamic Programming and Pairwise Sequence Alignment"**

Zahra Ebrahim zadeh

If you have any questions on these exercises, please contact me: z.ebrahimzadeh@utoronto.ca

1) Given an mRNA sequence= SQPTQYAWKW and the original sequence = DELSQPTQYAWKWVRETQ from the organism genome. We are interested in finding mRNA corresponding location in the organism genome .

a) Which alignment should we perform?
b) Use the appropriate algorithm for this alignment. Fill in the DP matrix and calculate the optimal alignment score. Obtain the optimal alignment by traceback the matrix.

2) In the class we saw to recover optimal alignment, arrows indicating optimal forward path are placed in matrix. Show how optimal alignment can be efficiently retrieved without these arrows.

3) In traceback of a pairwise sequence alignment, several optimal alignments can be obtained. Give a DP algorithm to compute the number of distinct co-optimal alignments.

4) Given two sequences with aligning score value of match = 2, mismatch = -1, gap = -1, which alignment score is larger: their local similarity or their global similarity? How does their semi-global similarity compare with the other two values?
Hint: Consider the scoring value in any alignment that two sequences would return.

5) Given the two following alignments:

Alignment 1:

Seq1 AGTGTGAAGGTCCCGGCTAAT---CG

Seq2 A-TGCG--GCTAATGGC-AATATACG

Alignment 2:

Seq1 AGTGTGAAGGTCCCGGCTAAT---------CG
Seq2 A---TG-------CGGCTAATGGCAATATACG

a) The two alignments above were created using the same gap opening penalty, but different gap extension penalties. What can you say about the relative values of the gap extension penalties in the alignments above?

b) Describe the alignment you would expect if the gap opening penalty was infinity. What kind of alignment would you expect if the gap start penalty and gap extension penalty were both zero?

6) In the class we saw global alignment using Needleman-Wunsch algorithm for linear gap model takes O(nm) time and O(nm) space.

a) Write an algorithm to compute global alignment in linear time. How big memory space does this algorithm use?
b) Write a DP algorithm to compute global alignment in linear space. What is the runtime of this algorithm?

7) As indicated in the class NW algorithm for general gap model (slide) takes O(mn2) where n>m.
Modify this algorithm so that it runs in O(nm) and O(nm) space.
Hint: use more than one matrix.

8) A palindromic sequence is a nucleic acid sequence that is the same whether read 5' to 3' on one strand or 5' to 3' on the complementary strand.

a) Write a DP algorithm to find the longest palindromic subsequence. Give the recurrence.
b) Analysis the runtime and space. Justify your answer.

9) Bacterial DNA is usually circular.

a) Write an efficient DP algorithm for global alignment of two circular sequences in less than cubic time.
b) Prove the correctness of your algorithm and justify the runtime.

BCB410 - Zahra Ebrahim zadeh

Exercises with solution - Dynamic Programming and Pairwise Sequence Alignment

If you have any questions on these exercises, please contact me: z.ebrahimzadeh@utoronto.ca

1) In the class we saw global alignment using Needleman-Wunsch algorithm for linear gap model takes O(nm) time and O(nm) space.

We may align very large sequences, where the algorithm using polynomial-space are not efficient. Modify a known alignment DP algorithms to compute optimal global alignment score in linear space i.e. O(min(m,n)) where m, n are length of two sequences. Justify the runtime.

Answer: We should note that the calculation of any entry of the matrix we saw in class only requires knowledge of the results from the current and previous rows i.e for any entry, its score obtained considering 3 neighbor entries as: diagonal entry, top entry, and left entry.
Therefore, we calculate our optimal values using a 2-dimentional array of 2x(min(m,n)). Then two rows would be $NW_{0,j}$ and $NW_{1,j}$. To fill the matrix, we first initialize row_0 according to the Needleman-Wunsch Algorithm base case. Then we fill the row_1 using the recurrence: $NW_{1,j} = \max\begin{Bmatrix} NW_{0,j-1} + s(v_i, w_j) \\ NW_{0,j} + g \\ NW_{1,j-1} + g \end{Bmatrix}$

Once all the entries of the NW1,j row are calculated, the value of each matrix entries of NW1,j is transferred to NW0,j. We keep computing the optimal values, at each step row_1 is equivalent of the i^{th} row in the full matrix.

In this method, as we store two rows of the matrix, the memory requirement is twice the length of the shorter sequence. Hence, that is O(min(m,n)).

2) As indicated in the class NW algorithm for general gap model (slide 22) takes $O(mn^2)$ where n>m. Modify this algorithm for affine gap model, so that it runs in O(nm) and O(nm) space. Hint: you can use more than one matrix.

Answer: The NW algorithm we saw was: $NW_{i,j} = \max\begin{Bmatrix} NW_{i-1,j-1} + s(v_i, w_j) \\ [NW_{i-n_{gap1},j} + g(n_{gap1})]_{1 \le n_{gap1} \le i} \\ [NW_{i,j-n_{gap2}} + g(n_{gap2})]_{1 \le n_{gap2} \le j} \end{Bmatrix}$

We need two matrices for the last two statement in order to reduce the runtime.

Hence, lets $R_{i,j} = \max\{NW_{i-n_{gap1},j} + g(n_{gap1})\}_{1 \le n_{gap1} \le i}$ (I) and that is equal to:

$$R_{i,j} = \max\begin{Bmatrix} NW_{i-1,j} + g(1) \\ [NW_{i-n_{gap1},j} + g(n_{gap1})]_{2 \le n_{gap1} \le i} \end{Bmatrix}$$

$$\downarrow$$

$$R_{i,j} = \max\begin{Bmatrix} NW_{i-1,j} + g(1) \\ [NW_{i-n_{gap1}-1,j} + g(n_{gap1}+1)]_{1 \le n_{gap1} \le i-1} \end{Bmatrix}$$

$$\downarrow$$

$$R_{i,j} = \max\begin{Bmatrix} NW_{i-1,j} - 1 \\ [NW_{i-n_{gap1}-1,j} + g(n_{gap1}) - (gap\_extension\_penalty)]_{1 \le n_{gap1} \le i-1} \end{Bmatrix}$$

$$\downarrow$$

$$R_{i,j} = \max\begin{Bmatrix} NW_{i-1,j} - (gap\_opening\_penalty) \\ [NW_{i-n_{gap1}-1,j} + g(n_{gap1})]_{1 \le n_{gap1} \le i-1} - (gap\_extention\_penalty) \end{Bmatrix}$$

Please note that here, we assume absolute values of gap opening penalty and gap extension penalty. Let gap_opening_penalty = O and gap_extension_penalty = E. Then from (I) substituting i-1 for i: $R_{i-1,j} = \max\{NW_{i-n_{gap1}-1,j} + g(n_{gap1})\}_{1 \le n_{gap1} \le i-1}$, the above formula can be written as:

$R_{i,j} = \max\begin{Bmatrix} NW_{i-1,j} - (O) \\ R_{i-1,j} - (E) \end{Bmatrix}$ This is recursive definition for aligning residues in sequence V with gaps in W.

The other case, aligning residues in sequence W with gaps in V, can be derived in similar way. Let other matrix be $L_{i,j}$, then

$$C_{i,j} = \max\begin{Bmatrix} NW_{i,j-1} - O \\ C_{i,j-1} - E \end{Bmatrix}$$

Therefore,

$$NW_{i,j} = \max\begin{Bmatrix} NW_{i-1,j-1} + s(v_i, w_j) \\ [NW_{i-n_{gap1},j} + g(n_{gap1})]_{1 \le n_{gap1} \le i} \\ [NW_{i,j-n_{gap2}} + g(n_{gap2})]_{1 \le n_{gap2} \le j} \end{Bmatrix} \rightarrow NW_{i,j} = \max\begin{Bmatrix} NW_{i-1,j-1} + s(v_i, w_j) \\ R_{i,j} \\ C_{i,j} \end{Bmatrix}$$

3) A palindromic sequence is a nucleic acid sequence that is the same whether read 5' to 3' on one strand or 5' to 3' on the complementary strand. Write a DP algorithm to find the longest palindromic subsequence. Give the recurrence and justify your answer.

Answer: We know that DNA is double stranded and a nucleic acid in one strand is complement of its opposite nucleic acid in other strand. Therefore, in order to determine palindromic sequence it is sufficient to consider one strand and we should note that palindromic sequence has to have an even length >= 2. Odd length subsequence cannot form palindromic DNA.
Is A palindrome? What about ACGCT? These are not palindrome.
T TGCGA

Let P be the nxn matrix such that P[i,j] store the length of palindromic subsequence $v_i,...,v_j$ of V, where |V|=n. Then we can drive the following recurrence:

$$P[i,j] = \begin{cases} P[i+1, j-1] + 2 & \text{if complement}(v_i, v_j) \\ -\infty & \text{if not complement}(v_i, v_j) \text{ or } i=j \\ 0 & \text{if } i > j \end{cases}$$

Using similar strategy as in traceback of global algorithm we can indicate the path to palindromic subsequences. After computing the matrix P, starting from the largest value in the matrix we get two initial bases of the longest palindrome and following the path we can then insert the next palindromic subsequence between the two outer bases of largest palindrome. We continue retrieving the palindrome in this way recursively.
The runtime is $O(n^2)$, as each entry of the matrix takes constant time to be computed.

4) Bacterial DNA is usually circular. Explain how we can use DP method for finding global alignment of two circular sequences V and W of length n and m respectively, for less than $O(n^2m^2)$ runtime.

Answer: Since the sequences are circular and we don't know where the beginning of either sequences is, we need to note that there are n circular shifts of V, and m circular shifts of W. Thus there are nm circular shifts. To determine an optimal alignment, we can

fix a start point on the longer sequence, i.e max(m,n), and perform standard global alignment for min(m,n) times, i.e for all shifts of smaller sequence.
The run time is O(nm.min(m,n)) since we would have min(m,n) matrices of size nxm and calculation of each entry would take O(1).

(An alternative way is constructing a 3-dimensional array of size n x m x min(m,n) and doing the global alignment for this matrix. We need to add a 3rd dimension [1...min(n,m)] to recurrence of global algorithm we saw in class. The runtime is O(nm.min(m,n).)

5) Consider the following two-player game: given an even number of amino acids with their molecular weight from a protein sequence, i.e. {a(1),...,a(n)}, where a(i) give the molecular weight of amino acid i. The game proceeds as follows: a player chooses to delete (and keep) a amino acid from either end of the sequence, and then it is the other player's turn to do the same. Play continues until all denominations have been removed.

Show how you can determine the maximum molecular weight of amino acids you are guaranteed to win if you take the first turn. Justify your answer.
Hint: Derive a recurrence M[i,j], for i <= j, indicating the maximum possible molecular weight you are guaranteed to win using amino acids {a(i),...,a(j)}.

Answer: Let M[i,i] for i<=j, be the maximum possible sum molecular weight we are guaranteed to win using remaining amino acids from i to j, where it is our turn to take. For M[i,j], there are two cases: we pick amino acid i, obtaining a(i), or we pick amino acid j, obtaining a(j) molecular weight.
Case 1: We pick a(i):

Subcase 1a. our opponent picks a(j). Then {a(i+1),...,a(j-1)} amino acids left.
Subcase 1b. our opponent picks a(i+1). Then {a(i+2),...a(j)} amino acids left.
Since the opponent is as smart as us, he would have chosen the choice that yields the minimum molecular weight sum to us. The guaranteed molecular weight in remaining amino acids would be the smallest of subcases. i.e. min{ M[i+1,j-1], M[i +2,j] }. Thus maximum amount we can get when we pick a(i) is :
Case1 = a(i) + min{ M[i+1,j-1],M[i+2,j] }

Case 2: We pick a(j):

Subcase 2a. our opponent picks a(i). Then {a(i+1),...,a(j-1)} amino acids left.
Subcase 2b. our opponent picks a(j-1). Then {a(i),...,a(j-2)} amino acids left.
The guaranteed molecular weight in remaining amino acids would be the smallest of the subcases. i.e. min{ M[i+1,j-1], M[i,j-2] }
Thus maximum amount we can get when we picked a(j) is :
Case2 = a(j) + min{M[i+1,j-1],M[i,j-2]}

Therefore, to maximize our profit, we choose max{ picking a(i)=Case1, picking a(j)

=Case2 }. Then:

$$M[i,j] = \max\begin{Bmatrix} a(i) + \min\{M[i+1, j-1], M[i+2, j]\} \\ a(j) + \min\{M[i+1, j-1], M[i, j-2]\} \end{Bmatrix}$$

6) Sometimes we are interested to find longest common subsequence (LCS) of two sequences, where the residues in LCS appear in both sequences in the same order but not necessarily consecutively. For example is V= CGGTCGAGTGCGCGGAAGCGC and W=GTCGTTCGGAATGCTT, then LCS=GTCGTCGGAAGC. Write an efficient algorithm to find the longest subsequence common to two sequences.

a) Give an optimal substructure of an LCS and prove its correctness.

Answer: First we characterize the optimal substructure of an LCS as follows:

Let V = (v1,v2,...,vn) and W= (w1,w2,...,wm) be two sequences, and Z = (z1,z2,..,zk) be any longest common subsequence of V and W. We can observe that an LCS of two sequences contains within it an LCS of prefixes of the two sequences.Then:

1) If $v_n = w_m$, then $z_k = v_n = w_m$ and Z_{k-1} is an LCS of V_{n-1} and W_{n-1}.
2) If $v_n \neq w_m$, then $z_k \neq v_n$ that means Z is an LCS of V_{n-1} and W.
3) If $v_n \neq w_m$, then $z_k \neq w_m$ that means Z is an LCS of V and W_{n-1}.

Proof by contradiction: Assume Z is LCS. For case 1, if $z_k \neq v_n$, then we could append $v_n = w_m$ to Z to obtain longer common subsequence of length k+1. This is contradicting that Z is LCS of V and W. Thus $z_k = v_n = w_m$. The prefix Z_{k-1} is the LCS for V_{n-1} and W_{n-1}. Suppose there exist a common subsequence X of V_{n-1} and W_{n-1} with length greater than k-1, appending $v_n = w_m$ to X yields a common subsequence with length greater than k, that is contradicting Z is LCS. For case 2, if there were a common subsequence X of Vn-1 and W with length greater than k, then X would be a LCS of Vn and W, that is contradicting that Z is an LCS of V and W. Case 3 has similar argument as 2.

b) Give the recurrence and justify your answer.

Answer: Construct the matrix L of size n+1 x m+1. Let L[i,j] be the length of an LCS of subsequence V_i and W_j. In characterized substructure of an LCS, we can see that for LCS we would have two major cases for when $v_i = w_i$ and $v_i \neq w_j$. For $v_i \neq w_j$ we have two subproblems of finding LCS for V_{n-1} and W and LCS for V and W_{n-1}.

Then:

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

c) Give the pseudocode of the algorithm and explain how to retrieve the LCS sequence from your algorithm.

Answer: We fill the matrix L in row major order, we also store paths we obtain the optimal values in a separate matrix P of size nxm.
Pseudocode will be:

```
LCS(V,W):
   for i=1 to |V|
       L[i,0]=0
   for j=0 to |W|
       L[0,j]=0
   for i =1 to |V|
        for j=1 to |W|
            if v_i == w_i
                  L[i,j] = L[i-1,j-1]+1
                P[i,j] = 0        // This is when the value of L[i,j[ comes from diagonal entry.
            else if L[i-1,j] >= L[i,j-1]
                     L[i,j] = L[i-1, j]
                     P[i,j] = 1     //This is when the value of L[i,j] comes from the top entry.
            else: L[i,j] = L[i, j-1]
                    P[i,j] = -1     //This is when the value of L[i,j] comes from the left entry.
```

We obtain LCS sequence from end of sequences, i.e starting P[n,m]. The pseudocode for getting LCS:

```
Retrieve_LCS (P, V, i, j):
   lcs_seq = “”
    if i = 0 or j = 0 return
    if P[i,j] == 0:
        Retrieve_LCS(P,V, i-1, j-1)
        lcs_seq = v_i + lcs_seq
    else if P[i,j] == 1:
        Retrieve_LCS(P, V, i-1, j)
    else: Retrieve_LCS(P, V, i, j-1)
```

d) What is the runtime of your algorithm. Justify your answer.

Answer: The LCS(V,W) takes nm steps as we have the i and j for loops. and Retrieve_LCS takes O(n+m) time as the value of i and/or j is decreases in each recursive call. Hence the algorithm will terminates once i or j become 0.
Thus the overall runtime is O(nm), since O(nm)+O(n+m) is O(nm).

7) Given two sequences with aligning score value of match = 2, mismatch = -1, gap = -1, which alignment score is larger: their local similarity or their global similarity? How does their semi-global similarity compare with the other two values?

Answer: With this scoring where value(gap) $\leq$ value(mismatch) < value(match), we have value of optimal_Global $\leq$ optimal_Semiglobal $\leq$optimal_Local.
A semiglobal alignment will have at least the global score because it could find the same alignment but would not be penalized for gaps on the sides of sequences. Similarly, local alignment score is at least as semi-global score since the optimal score of the alignment is not reduced by any net-negative end of the alignment and pairing of residues that may reduce its score.
In other words, with this scoring every global alignment is a semiglobal alignment and every semiglobal alignment is a local alignment.