CS 238, Spring 2018
Homework #**03**

Student Name: **Dang Tu Nguyen**
Student ID: **861309226**

Problem 1. A divide-and-conquer algorithm using space-saving technique for three sequence alignment.

Problem: Given three sequences s_1, s_2, and s_3 of length m, n, and l, respectively. Let $S[i, j, k]$ be the optimal score of the global alignment of three prefixes $s_1(0, i)$, $s_2(0, j)$, and $s_3(0, k)$. Our problem is to compute $S[m, n, l]$ and the three aligned sequences (the optimal path).

Basic Idea: First, we will compute the intersection I of the optimal path $P_{(0,0,0)\to(m,n,l)}$ and the middle plane $M\{-, -, l/2\}$. To do this, we need to compute the optimal forward score table of three sequences from the top to $M\{-, -, l/2\}$, and compute the optimal backward score table of the three reversed sequences from bottom to $M\{-, -, l/2\}$. $I(i_M, j_M, l/2)$ will be the point in $M\{-, -, l/2\}$ whose sum of forward score and backward score is maximum. This helps to divide the problem of finding the optimal path $(0, 0, 0) \to (m, n, l)$ into two sub-problems of finding two optimal paths $(0, 0, 0) \to (i_M, j_M, l/2)$ and $(i_M, j_M, l/2) \to (m, n, l)$. We can recursively solve the sub-problems. The base case would be when one of the sequences has one or zero letter. Then, we use the original $O(mnl)$ dynamic programming algorithm for three sequences alignment.

Time Complexity: The total size of 2 sub-problems equals to 1/4 size of original problem. Similarly, the total size of 4 sub-problems equals to 1/16 size of original problem. Let T be the size of the original problem, the total size of all recursive problems is at most $T + T/4 + T/16 + ... = 4T/3$. Therefore, the running time of this algorithm is still $O(mnl)$.

Recurrent formula to calculate the optimal 3D scores:

- **Two sequence alignment with u and v**: add insertion penalty score for the third sequence. These 2D score arrays are used to initialize 3D score matrix.
  Initialization:
  $S_{0,0} = 0$
  $S_{i,0} = S_{i-1,0} + 2 * \delta(u_i, -)$
  $S_{0,j} = S_{0,j-1} + 2 * \delta(v_j, -)$

Recurrence formula:

$$S_{i,j} = max \begin{cases} S_{i-1,j} + 2 * \delta(u_i, -) \\ S_{i,j-1} + 2 * \delta(v_j, -) \\ S_{i-1,j-1} + \delta(u_i, v_i) + \delta(u_i, -) + \delta(v_j, -) \end{cases}$$

- **Three sequence alignment with u, v and w:**
  Initialization: Use the above 2D recurrence to compute initial matrixes in 3D.
  Recurrence formula:

$$S_{i,j,k} = max \begin{cases} S_{i-1,j,k} + 2 * \delta(u_i, -) \\ S_{i,j-1,k} + 2 * \delta(v_j, -) \\ S_{i,j,k-1} + 2 * \delta(w_k, -) \\ S_{i-1,j-1,k} + \delta(u_i, v_j) + \delta(u_i, -) + \delta(v_j, -) \\ S_{i-1,j,k-1} + \delta(u_i, w_k) + \delta(u_i, -) + \delta(w_k, -) \\ S_{i,j-1,k-1} + \delta(v_j, w_k) + \delta(v_j, -) + \delta(w_k, -) \\ S_{i-1,j-1,k-1} + \delta(u_i, v_j) + \delta(u_i, w_k) + \delta(v_j, w_k) \end{cases}$$

$\delta(x, y)$ is the alignment score between x and y, which can be obtained from the 5x5 input matrix.

Algorithm 1: Algorithm 1 reveals how we use recursive divide and conquer approach to find the optimal path and score.

Algorithm 2: Compute optimal forward score table from $(s_1\_start, s_2\_start, s_3\_start)$ to $(s_1\_end, s_2\_end, s_3\_end)$.

First, we compute optimal 2D alignment scores s_{12}_scores (first and second sub-sequences), s_{31}_scores (third and first sub-sequences), and s_{32}_scores (third and second sub-sequences) using the 2D recurrence formula in section 6.6, page 177 of the textbook. Then, we use a 2D matrix $forward\_scores$ where $forward\_scores(i, j)$ is the optimal score of the path from $(s_1\_start, s_2\_start, s_3\_start)$ to (i, j, k). To calculate the score of a point in the $forward\_scores$, we need the scores of 7 neighboring points, in which 3 points in the same plane as the current point ($\{-, -, k\}$) and 4 points in the previous plane ($\{-, -, k-1\}$). In other words, we only need to store the most recent plane and the current plane in two 2D arrays to calculate the score of the current point. Therefore, all memory we need is $mn(1 + 1/4 + 1/16 + ...) = 4mn/3 = O(mn)$.

To compute the backward score table, we first revert the three sequences and use algorithm 2 to compute the forward score table for the reverted sequences. Then, we revert the result forward score table 180 degree to get the backward score table.

Note: Source code is attached to the end of the report.

Algorithm 1 Find the optimal score and path from $(s_1\_start, s_2\_start, s_3\_start)$ to $(s_1\_end, s_2\_end, s_3\_end)$

procedure RECUR_ALIGN($s_1\_start, s_2\_start, s_3\_start, s_1\_end, s_2\_end, s_3\_end$)
  if $s_1\_start + 1 \geq s_1\_end$ or $s_2\_start + 1 \geq s_2\_end$ or $s_3\_start + 1 \geq s_3\_end$ **then**
    DP_ALIGN($s_1\_start, s_2\_start, s_3\_start, s_1\_end, s_2\_end, s_3\_end$) ▷ Use original dynamic programming for a base case
  else
    $mid \leftarrow (s_3\_start + s_3\_end)/2$
    $forward\_scores \leftarrow$ Compute forward 3D scores for three sub-sequences $(s_1\_start, s_2\_start, s_3\_start, s_1\_end, s_2\_end, mid)$
    $backward\_scores \leftarrow$ Compute backward 3D scores for three reversed sub-sequences $(s_1\_start, s_2\_start, mid, s_1\_end, s_2\_end, s_3\_end)$
    $i_M, j_M \leftarrow$ Find max sum of $(forward\_scores, backward\_scores)$
    RECUR_ALIGN($s_1\_start, s_2\_start, s_3\_start, i_M, j_M, mid$)
    RECUR_ALIGN($i_M, j_M, mid, s_1\_end, s_2\_end, s_3\_end$)
  end if
end procedure

Experiment Results: The experiments were run in a MacBook laptop with Processor 2.9 GHz Intel Core i5 and 16 GB 1867 MHz DDR3. The results are shown in Table 1. There are many conserved regions, mostly in the middle of the sequences. Moreover, most of the mutations appear at the beginning or the end of the sequences.

Table 1: Experiment results.

Dataset	Score	Length of align.	Num. of exact matches	Running time	Memory
1	4752	631	427	40 *seconds*	19.76 MB
2	4000	2720	1403	27.95 *minutes*	67.37 MB
3	928	4904	1917	3.89 *hours*	24.09 MB
4	62770	6398	4787	12.76 *hours*	1599.83 MB

Problem 2. Recurrence relations for multiple alignment of 4 sequences o, u, v, and w

Algorithm 2 Compute optimal forward score table from $(s_1\_start, s_2\_start, s_3\_start)$ to $(s_1\_end, s_2\_end, s_3\_end)$

procedure COMPUTE_FORWARD_3D_SCORES($s_1\_start, s_2\_start, s_3\_start, s_1\_end, s_2\_end, s_3\_end$)
    $s_{12}\_scores \leftarrow$ Compute 2D forward scores of the first and second sub-sequences $(s_1, s_2, s_1\_start, s_2\_start, s_1\_end, s_2\_end)$
    $s_{31}\_scores \leftarrow$ Compute 2D forward scores of the third and first sub-sequences $(s_3, s_1, s_3\_start, s_1\_start, s_3\_end, s_1\_end)$
    $s_{32}\_scores \leftarrow$ Compute 2D forward scores of the third and second sub-sequences $(s_3, s_2, s_3\_start, s_2\_start, s_3\_end, s_2\_end)$
    $forward\_scores \leftarrow s_{12}\_scores$
    for $k = 1$ to $s_3\_end - s_3\_start$ **do**
        Update $forward\_scores$ based on s_{31}_scores and s_{32}_scores
        for $i = 1$ to $s_1\_end - s_1\_start$ **do**
            for $j = 1$ to $s_2\_end - s_2\_start$ **do**
                $forward\_scores[i][j] \leftarrow$ Compute scores of 7 cases and get the max one
            end for
        end for
    end for
    return $forward\_scores$
end procedure

under the SP (sum-of-pairs) scoring rule.

$$S_{h,i,j,k} = max \begin{cases} S_{h-1,i,j,k} + \delta(o_h, -, -, -) \\ S_{h,i-1,j,k} + \delta(-, u_i, -, -) \\ S_{h,i,j-1,k} + \delta(-, -, v_j, -) \\ S_{h,i,j,k-1} + \delta(-, -, -, w_k) \\ S_{h-1,i-1,j,k} + \delta(o_h, u_i, -, -) \\ S_{h-1,i,j-1,k} + \delta(o_h, -, v_j, -) \\ S_{h-1,i,j,k-1} + \delta(`o_h, -, -, w_k) \\ S_{h,i-1,j-1,k} + \delta(-, u_i, v_j, -) \\ S_{h,i-1,j,k-1} + \delta(-, u_i, -, w_k) \\ S_{h,i,j-1,k-1} + \delta(-, -, v_j, w_k) \\ S_{h-1,i-1,j-1,k} + \delta(o_h, u_i, v_j, -) \\ S_{h-1,i-1,j,k-1} + \delta(o_h, u_i, -, w_k) \\ S_{h-1,i,j-1,k-1} + \delta(o_h, -, v_j, w_k) \\ S_{h,i-1,j-1,k-1} + \delta(-, u_i, v_j, w_k) \\ S_{h-1,i-1,j-1,k-1} + \delta(o_h, u_i, v_j, w_k) \end{cases}$$

or

$$S_{h,i,j,k} = max \begin{cases} S_{h-1,i,j,k} + 3 * \delta(o_h, -) + 3 * \delta(-, -) \\ S_{h,i-1,j,k} + 3 * \delta(u_i, -) + 3 * \delta(-, -) \\ S_{h,i,j-1,k} + 3 * \delta(v_j, -) + 3 * \delta(-, -) \\ S_{h,i,j,k-1} + 3 * \delta(w_k, -) + 3 * \delta(-, -) \\ S_{h-1,i-1,j,k} + 2 * \delta(o_h, -) + 2 * \delta(u_i, -) + \delta(o_h, u_i) + \delta(-, -) \\ S_{h-1,i,j-1,k} + 2 * \delta(o_h, -) + 2 * \delta(v_j, -) + \delta(o_h, v_j) + \delta(-, -) \\ S_{h-1,i,j,k-1} + 2 * \delta(o_h, -) + 2 * \delta(w_k, -) + \delta(o_h, w_k) + \delta(-, -) \\ S_{h,i-1,j-1,k} + 2 * \delta(u_i, -) + 2 * \delta(v_j, -) + \delta(u_i, v_j) + \delta(-, -) \\ S_{h,i-1,j,k-1} + 2 * \delta(u_i, -) + 2 * \delta(w_k, -) + \delta(u_i, w_k) + \delta(-, -) \\ S_{h,i,j-1,k-1} + 2 * \delta(v_j, -) + 2 * \delta(w_k, -) + \delta(v_j, w_k) + \delta(-, -) \\ S_{h-1,i-1,j-1,k} + \delta(o_h, u_i) + \delta(o_h, v_j) + \delta(o_h, -) + \delta(u_i, v_j) + \delta(u_i, -) + \delta(v_j, -) \\ S_{h-1,i-1,j,k-1} + \delta(o_h, u_i) + \delta(o_h, -) + \delta(o_h, w_k) + \delta(u_i, -) + \delta(u_i, w_k) + \delta(-, w_k) \\ S_{h-1,i,j-1,k-1} + \delta(o_h, -) + \delta(o_h, v_j) + \delta(o_h, w_k) + \delta(-, v_j) + \delta(-, w_k) + \delta(v_j, w_k) \\ S_{h,i-1,j-1,k-1} + \delta(-, u_i) + \delta(-, v_j) + \delta(-, w_k) + \delta(u_i, v_j) + \delta(u_i, w_k) + \delta(v_j, w_k) \\ S_{h-1,i-1,j-1,k-1} + \delta(o_h, u_i) + \delta(o_h, v_j) + \delta(o_h, w_k) + \delta(u_i, v_j) + \delta(u_i, w_k) + \delta(v_j, w_k) \end{cases}$$

Problem 3.

Let consider the quartet A, B, C, D:

$$\begin{cases} d_{AB} + d_{CE} = 14 \\ d_{AC} + d_{BD} = 12 \\ d_{AD} + d_{BC} = 14 \end{cases}$$

From the above equations, we have the following tree:

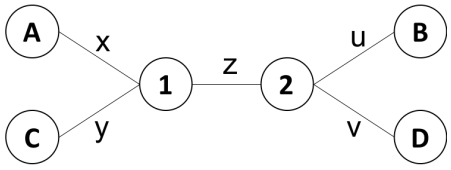

Figure 1: Evolution tree of the quartet A, B, C, D.

Let consider the quartet A, B, C, E:

$$\begin{cases} d_{AB} + d_{CE} = 17 \\ d_{AC} + d_{BE} = 15 \\ d_{AE} + d_{BC} = 17 \end{cases}$$

From the above equations, we can insert E into either of the following three places: between 2 and B, between 2 and D, between 1 and 2.
Let consider the quartet A, B, D, E:

$$\begin{cases} d_{AB} + d_{DE} = 12 \\ d_{AD} + d_{BE} = 14 \\ d_{AE} + d_{BD} = 14 \end{cases}$$

From the above equations, we can insert E into only one position between 2 and D.

From Figure 1 and the distance matrix, we have the following:

$$\begin{cases} x + y = 7 \\ x + z + u = 5 \\ x + z + v = 6 \\ u + v = 5 \\ y + z + u = 8 \\ y + z + v = 9 \end{cases}$$

Solution:

$$\begin{cases} x = 2 \\ y = 5 \\ z = 1 \\ u = 2 \\ v = 3 \end{cases}$$

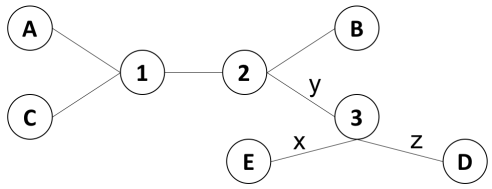

Figure 2: Evolution tree of A, B, C, D, E.

From Figure 2 and the distance matrix, we have the following:

$$\begin{cases} y + z = 3 \\ x + z = 7 \\ x + y = 6 \end{cases}$$

Solution:

$$\begin{cases} x = 5 \\ y = 1 \\ z = 2 \end{cases}$$

Thus, we have the final evolution tree with lengths of branches as shown in Figure 3.

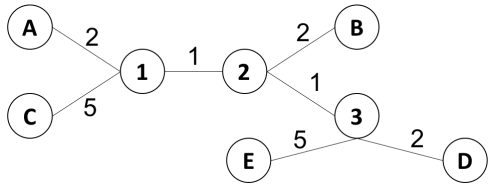

Figure 3: Evolution tree of A, B, C, D, E with lengths of branches.