Algoritmer for bioinformatikk - TDT4287 - Øving 1

Ola Kristoffer Hoff

22. September 2022

0 String terms

Given string ω = ABBACBDACC, and a set of strings $\mathbb{S}$ = {ACCB, AB, BCDA, CBDA, ACC, BACA, ABBA, ω}

0.1 Which of the strings from $\mathbb{S}$ are substrings of ω?

{AB, CBDA, ACC, ABBA, ω}

0.2 Which of the strings from $\mathbb{S}$ are prefixes of ω?

{AB, ABBA, ω}

0.3 Which of the strings from $\mathbb{S}$ are suffixes of ω?

{ACC, ω}

0.4 Which of the strings from $\mathbb{S}$ are subsequences of ω?

{AB, BCDA, CBDA, ACC, BACA, ABBA, ω}

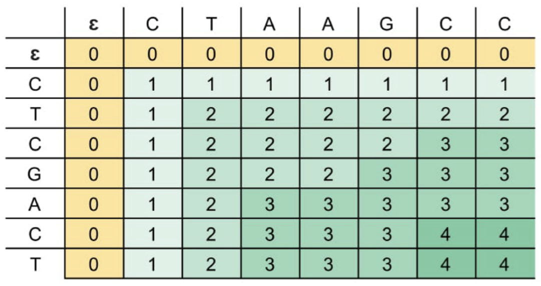
1 Longest common subsequence (LCS)

1.1 Alignments

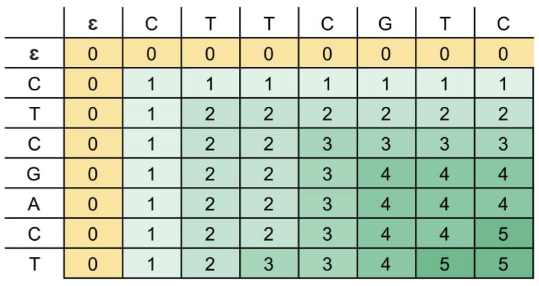
1.1.1 Compute the LCS for the following pair of strings: ω_1 = ACGGTAC ω_2 = CTCGACT

+	¢	A	C	G	G	T	A	C
¢	0	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1	1
T	0	0	1	1	1	2	2	2
C	0	0	1	1	1	2	2	3
G	0	0	1	2	2	2	2	3
A	0	1	1	2	2	2	3	3
C	0	1	2	2	2	2	3	4
T	0	1	2	2	2	3	3	4

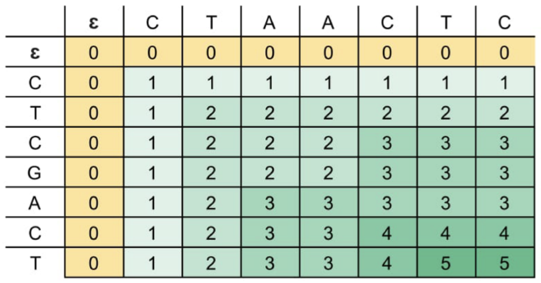
1.1.2 Determine the alignments, given the following dynamic programming (DP) tables:



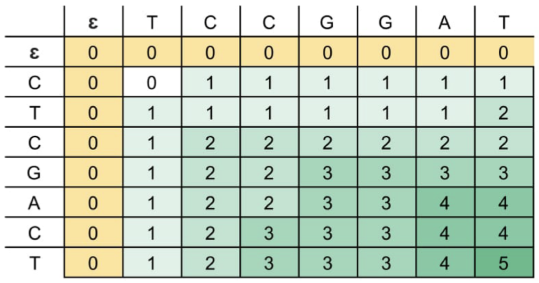
	ε	C	T	A	A	G	C	C
ε	0	0	0	0	0	0	0	0
C	0	1	1	1	1	1	1	1
T	0	1	2	2	2	2	2	2
C	0	1	2	2	2	2	3	3
G	0	1	2	2	2	3	3	3
A	0	1	2	3	3	3	3	3
C	0	1	2	3	3	3	4	4
T	0	1	2	3	3	3	4	4



	ε	C	T	T	C	G	T	C
ε	0	0	0	0	0	0	0	0
C	0	1	1	1	1	1	1	1
T	0	1	2	2	2	2	2	2
C	0	1	2	2	3	3	3	3
G	0	1	2	2	3	4	4	4
A	0	1	2	2	3	4	4	4
C	0	1	2	2	3	4	4	5
T	0	1	2	3	3	4	5	5



	ε	C	T	A	A	C	T	C
ε	0	0	0	0	0	0	0	0
C	0	1	1	1	1	1	1	1
T	0	1	2	2	2	2	2	2
C	0	1	2	2	2	3	3	3
G	0	1	2	2	2	3	3	3
A	0	1	2	3	3	3	3	3
C	0	1	2	3	3	4	4	4
T	0	1	2	3	3	4	5	5



	ε	T	C	C	G	G	A	T
ε	0	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1	1
T	0	1	1	1	1	1	1	2
C	0	1	2	2	2	2	2	2
G	0	1	2	2	3	3	3	3
A	0	1	2	2	3	3	4	4
C	0	1	2	3	3	3	4	4
T	0	1	2	3	3	3	4	5

CTAA-G-CC-
CT–CGAC-T

CTTCGT-C-
CT-CG-ACT

CT–AACTC
CTCGA-CT-

-TCCGGA-T
CTC-G-ACT

1.1.3 How do we represent an alignment?

As we did in the previous task. We insert "-" into the strings where there has been an insertion/deletion.

1.2 LCS as a graph problem

1.2.1 Specify the edge set E. Determine |E|.

E contains edges that connect M[i, j] with M[i+1, j], M[i, j+1] and M[i+1, j+1], where $i \leq m-1$ and $j \leq n-1$.
$|E| = (m * n * 3) - 2((m-1) + (n-1) + 3) = 3mn - 2m - 2n + 1$.

1.2.2 Is it the shortest or longest path that we seek as the solution?

We want to find the shoetest path through the graph. This involves the highest number of diagonal paths, hens the most matches.

1.2.3 What does the number at table entry M[i, j] resemble for the aforementioned graph problem?

The LCS solution from M[0, 0] to M[i, j].

1.3 Backtracking the LCS

1.3.1 Fill in the code for line 10, so that the algorithm works correctly.

$10 : i = i - 1; j = j - 1;$

1.3.2 Prove that the algorithm always outputs the correct alignment.

It reverses the steps taken to find the LCS, hens it is correct.

1.4 *Alternative recurrence relation

1.4.1 Could the following recurrence relation be used as basis for the LCS algorithm? Provide reasoning.

$$\text{LCS}(S_i, R_j) = \begin{cases} \max\{LCS(S_{i-1}, R_j), LCS(S_i, R_{j-1})\} & \text{if } s_i \neq r_j \\ \min\{LCS(S_{i-1}, R_j), LCS(S_i, R_{j-1})\} + 1 & \text{if } s_i = r_j \end{cases}$$

When we take the other LCS we take the Max of the LCS's in the first case and the $LCS(S_{i-1}, R_{j-1}) + f(s_i, r_j)$. What happens then is that no value in M for any pair of $i \leq current\_i$ and $j \leq current\_j$ excluding $i = current\_i, j = current\_j$. So M[i, j] has to be the max value of all the values before it including itself. Therefore if the new value does not have a match, it must be equal to one of the other values, up or left, since these together form the max of all previous values. So the first half sounds good. If we frame the second one a bit differant, as to when can it not be true. It can only be false if up and left has value N and diagonal has value $< N$, since it then should be assigned to N given the other implementation of LCS. But fo this to happen we would have to have a match in $M[i, j]$ and since up and left has greater values than diagonal, they must have had a match earlier in their respective column and row. However since diagonl is the max of all values before it, the values inside the rectangle formed by taking diagonals j and i value and drawing up the column and row. For an earlier match in the row and column for left and up values we need another occurences of their match at $M[i, j]$, lets say the letter: 'A'. If there is an earlier 'A' in both the column and the row, then these two values must have had a match which would correspond to the increase the row and column for the up and left values got earlier. But this value lays within the area of which diagonal is based on, thus it cannot be the case that up and left can have greater values than diagonal when there is a match. So if the only case where the second expression would be false, is not a valid state, then the expression must be true.

2 Recursion vs Dynamic Programming

2.1 What is the difference between the recurrent and dynamic programming algorithm?

The recursion will recalculate every branch multiple times and have more memory usage. The DP solution stores the already calculated sub-solution and reuses the answer when needed.

2.2 Illustrate the difference by quantifying the number of comparisons made by recurrent and DP algorithm for strings of length 7. (Calculating max of three numbers takes 2 comparisons.)

The DP will fill out each matrix entry once, there are $N * N$ number of entries where each takes two comparisons, hens there is $N * N * 2$ comparisons. Given $N = 7$ there will be $7 * 7 * 2 = 98$ comparisons. With the recursive solution there will be 48638.

2.3 Is the recursive algorithm polynomial in time/space?

The recursive algorithm is exponential in time. Runs in O(n!), I think.

2.4 Dynamic programming schema

The schema hereunder is a general description of a dynamic programming algorithm for problem P.

I) Subproblems $\quad \mathcal{S} = \{\mathcal{L}(i)\}$

II) Direct computation of base cases $\quad \Omega(\mathcal{S}) \subseteq \mathcal{S}$

III) Recurrent computation of $\mathcal{L}(i) \in S \setminus \Omega(S)$

IV) Order in which $\mathcal{S}$ can be solved.

V) Solve $\mathcal{P}$ using solutions of subproblems

2.4.1 Identify phases I–V for the LCS algorithm.

I) The solution is based on the set of subproblems.
II) We need to initialise the DP-table with the basecases.
III) We find a given subproblem in the solution or from the basecase.
IV) We have to solve the subproblems in the correct order so that when the subproblems solution is needed it is already calculated.
V) End up with the solution to the problem from the solutions of the subproblems.

2.4.2 The order in IV has to be topological ordering w.r.t. the graph described in Lemma 1. Show why the ordering used in LCS is topological, and show an example of an ordering that is not topological.

We only connect nodes down right and diagonal down-right, hens no loops and recursive dependencies, hens a topological graph.

3 Edit Distance

Hamming distance is defined as $dH(\omega_1, \omega_2) = |\{i|\omega_1[i] \neq \omega_2[i]\}|$ Recall from lecture 2, that we defined Edit Distance (ED, or d_{ED}) as follows:

$$\text{ED}(S_i, R_j) = \min \begin{cases} ED(S_{i-1}, R_j) + 1 \\ ED(S_i, R_{j-1}) + 1 \\ ED(S_{i-1}, R_{j-1}) + f(s_i, r_j) \end{cases} \qquad \text{f(s, r)} = \begin{cases} 0 & r = s \\ 1 & \text{otherwise} \end{cases}$$

3.1 Fill in the correct inequality sign for the following expression: $d_H(\omega_1, \omega_2)$? $d_{ED}(\omega_1, \omega_2)$

$d_H(\omega_1, \omega_2) \neq d_{ED}(\omega_1, \omega_2)$

3.2 Can you reformulate the problem as a graph problem, just as we did for LCS in Lemma 1?

It is the exact same problem, only that we now have more edges and nodes to represent the differant tables and connections between them.

3.3 Show that d_{ED} is a distance (metric) in mathematical sense. That is, show that the following conditions are each satisfied for any x, y :

I) $d_{ED}(x, y) \geq 0$
II) $d_{ED}(x, y) = 0 \Leftrightarrow x = y$
III) $d_{ED}(x, y) = d_{ED}(y, x)$
IV) $d_{ED}(x, z) \leq d_{ED}(x, y) + d_{ED}(y, z)$

I) If they are equal then 0, else we have to do changes hens greater than 0.

II) If we change nothing they must be equal.
III) If we can edit x to y in N steps, then we can reverse the steps and apply them to y to get x.
IV) If y and z are equal the second step is zero so the expression is equal. But if they are different then, since edit distance is the fewest steps necessary to change, then doing x to y then y to z must be longer then going directly from x to z, otherwise the left hand side would be an invalid edit distance.

3.4 Is LCS : $\sum * \times \sum * \to \mathbb{R}$ also a distance (metric)?

If the value is higher it means there is more overlap between the strings. Whereas a lower number would mean few matches.

4 Local and Global alignment

Recall the recurrent relations for Global Alignment (GA) and Local Alignment (LA).

$$\text{GA}(S_i, R_j) = \max \begin{cases} LA(S_{i-1}, R_j) + I \\ LA(S_i, R_{j-1}) + I \\ LA(S_{i-1}, R_{j-1}) + \delta(s_i, r_j) \end{cases}$$

$$\text{LA}(S_i, R_j) = \max \begin{cases} LA(S_{i-1}, R_j) + I \\ LA(S_i, R_{j-1}) + I \\ LA(S_{i-1}, R_{j-1}) + \delta(s_i, r_j) \\ 0 \end{cases}$$

4.1 Compute GA and LA for sequences $\omega_1 = CTCTAGC$, $\omega_2 = CGGATAC$, with match score 1, mismatch -2, and indel penalty -2.

+	¢	C	T	C	T	A	G	C
¢	0	-1	-2	-3	-4	-5	-6	-7
C	-1	1	-1	-1	-3	-5	-7	-5
G	-2	-1	-1	-3	-3	-5	-4	-6
G	-3	-3	-3	-3	-5	-5	-4	-6
A	-4	-5	-5	-5	-5	-4	-6	-6
T	-5	-6	-4	-6	-4	-6	-6	-8
A	-6	-7	-6	-6	-6	-3	-5	-7
C	-7	-5	-7	-5	-7	-5	-5	-4

Table 1: GA table

+	¢	C	T	C	T	A	G	C
¢	0	0	0	0	0	0	0	0
C	0	1	0	1	0	0	0	1
G	0	0	0	0	0	0	1	0
G	0	0	0	0	0	0	1	0
A	0	0	0	0	0	1	0	0
T	0	0	1	0	1	0	0	0
A	0	0	0	0	0	2	0	0
C	0	1	0	1	0	0	0	1

Table 2: LA table

4.2 Compute GA with affine gaps for ω_1, ω_2, with match score 1, mismatch -2, gap open penalty -3, and gap extension penalty -1.

+	¢	C	T	C	T	A	G	C
¢	0	-1	-2	-3	-4	-5	-6	-7
C	-1	1	-3	-1	-5	-6	-7	-5
G	-2	-3	-1	-5	-3	-7	-5	-9
G	-3	-4	-5	-3	-7	-5	-6	-7
A	-4	-5	-6	-7	-5	-6	-7	-8
T	-5	-6	-4	-8	-6	-7	-8	-9
A	-6	-7	-8	-6	-10	-5	-9	-10
C	-7	-5	-9	-7	-8	-9	-7	-8

Table 3: GA affine table

+	¢	C	T	C	T	A	G	C
¢	0	-1	-2	-3	-4	-5	-6	-7
C	-1	-2	-3	-4	-5	-6	-7	-8
G	-2	-3	-4	-5	-6	-7	-8	-9
G	-3	-4	-5	-6	-7	-8	-9	-10
A	-4	-5	-6	-7	-8	-9	-10	-11
T	-5	-6	-7	-8	-9	-10	-11	-12
A	-6	-7	-8	-9	-10	-11	-12	-13
C	-7	-8	-9	-10	-11	-9	-13	-14

Table 4: GA_I affine table

+	¢	C	T	C	T	A	G	C
¢	0	-1	-2	-3	-4	-5	-6	-7
C	-1	-2	-3	-4	-5	-6	-7	-8
G	-2	-3	-4	-5	-6	-7	-8	-9
G	-3	-4	-5	-6	-7	-8	-9	-10
A	-4	-5	-6	-7	-8	-9	-10	-11
T	-5	-6	-7	-8	-9	-10	-11	-12
A	-6	-7	-8	-9	-10	-11	-9	-10
C	-7	-8	-9	-10	-11	-12	-13	-11

Table 5: GA_D affine table

4.3 Fill in the correct inequality: GA(S_i, R_j) ? LA(S_i, R_j)

It depends in most cases they are not equal, but there is possible for the global alignment to be the best alignment too. E.g. when S and R is equal.

4.4 Interpret the GA, and LA problems as graph problems again, alike in Lemma 1.

Just as before, it is the same problem, we only need to change the initial conditions. As we do with the first row and column in the DP-table.

4.5 Create a backtracking algorithm for GA with affine gaps, that derives the correct alignment from the filled DP matrices. We require that, alike the algorithm in 1.3, the algorithm does not compare sequence letters (and thus only reads the matrix values).

Just look at where the value came from, in the max evaluation, then go to that value and do the same till you have propagated to the start. In the LA case start at the highest value in the GA table, and backtrack till you encounter a zero value.