Lab 6. DNA Sequence data analysis

1. Objective

- To understand the sequence alignment
- To understand the dynamic programming
- To practice the C++ programming

2. Background

2.1 Sequence alignment and dynamic programming

Sequence alignment is the procedure of comparing two (pair-wise alignment) or more (multiple sequence alignment) sequences by searching for a series of individual characters or character patterns that are in the same order in the sequences. Sequence alignment is useful for discovering functional, structural, and evolutionary information in biological sequences. Similar sequences probably have the same function, such as a regulatory role in the case of similar DNA molecules, or a similar biochemical function and three-dimensional structure in the case of proteins. Additionally, if two sequences from different organisms are similar, there may have been a common ancestor sequence, and then the sequences are defined as being homologous.

One of the popular methods for sequence alignment is using dynamic programming (DP) algorithms. The dynamic programming method allows us to find the optimal alignment by recursively solving sub-problems. It is also guaranteed in a mathematical sense to provide the optimal (very best or highest-scoring) alignment for a given set of user-defined variables, including choice of scoring matrix and gap penalties. The first use of dynamic programming for comparing biological sequences was Needleman and Wunsch in 1970.

2.2 Practical application of DP on a biological problem: pairwise DNA sequence alignment

Assume that we have two DNA or Protein sequences, and we want to infer if they are homologous or not. To do this, we will calculate a score that reflects how similar the two sequences are (that is how likely they are to be derived from a common ancestor). Because sequences differ not just by substitution, but also by insertion and deletion, we want to optimally align the two sequences to maximize their similarity score. Now the problem is to align the residues (DNA nucleotides or Amino acids in Protein).

Let's set up the problem with some notations to solve it by dynamic programming. Call the two sequences x and y. They are of length M and N residues, respectively. The i^{th} residues in x is x_i and the j^{th} residue of y is y_j. For example, we have the following two DNA sequences, and want to know which alignment of residues gives high similarity (figure 1).

- X = TTCATA
- Y = TGCTCGTA

We will use a scoring system $\sigma(x_i, y_j)$ for aligning two residues (x_i, y_j) to each other and a gap penalty γ for every time we introduce a gap character. In general, the gap is represented by hyphen character "-".

We represent the score of two residues x_i in x and y_j in y by $\sigma(x_i, y_j)$, and that of two sequences x and y by $\sigma(x, y)$. For example, we have a scoring system of

+5 for a match,

-2 for a mismatch, and

-6 for each gap(insertion or deletion),

that is $\sigma(T,T) = \sigma(G,G) = +5$, $\sigma(C,G) = \sigma(T,A) = -2$, gap penalty $\gamma = -6$.

The following is one possible alignment with score -3 using the gap character.

X = T – T C – A T A

Y = T G C T C G T A

Score = +5 -6 -2 -2 -6 -2 +5 +5 = -3

With appropriate assumptions given as above examples, dynamic programming is guaranteed to give you a mathematically optimal solution. A Dynamic programming (DP) algorithm consists of four parts:

- recursive definition of the optimal score;
- matrix for remembering optimal scores of sub-problems;
- bottom-up approach of filling the matrix by solving the smallest sub-problem first;
- traceback of the matrix to recover the structure of the optimal solution that gave the optimal score.

For pairwise alignment, those steps are following.

2.2.1 Recursive definition of the optimal alignment score

There are only three ways the alignment can possibly end:

- residues x_M and y_N are aligned to each other;
- residues x_M is aligned to a gap character, and y_N appeared somewhere earlier in the alignment;
- or residue y_N is aligned to a gap character, and x_M appeared earlier in the alignment.

The optimal alignment will be the highest scoring of these three cases.

Crucially, we can recursively generalize this for the preceding subsequences of $x_1 \ldots x_M$ and $y_1 \ldots y_N$. Let S(i, j) be the score of the optimum alignment of subsequence $x_1 \ldots x_i$ to subsequence $y_1 \ldots y_j$. The

score is made in each case above:

- $S(i, j) = \sigma(x_i, y_j) +$ $S(i-1, j-1)$ for an optimal alignment up to this point,
- $S(i, j) =$ gap penalty $+ S(i-1, j)$
- $S(i, j) =$ gap penalty $+ S(i, j-1)$

This works because the problem breaks into independently optimizable pieces, as the scoring systems is strictly local to one aligned column at a time. That is, for instance, the optimal alignment of $x_1 \ldots x_i$ to $y_1 \ldots y_j$ is unaffected by adding on the aligned residue pair (x_i, y_j), and likewise, the score $\sigma(x_i, y_j)$ we add on is independent of the previous optimal alignment. So to calculate the score of the three cases, we will need to know three more alignment scores for the smaller problems:

$S(i-1,j-1)$, $S(i-1,j)$, $S(i,j-1)$

And to calculate those, we need the solutions for nine small problems:

- for $S(i-1,j-1)$: $S(i-2, j-2)$, $S(i-2, j-1)$, $S(i-1, j-2)$,
- for $S(i-1,j)$: $S(i-2, j-1)$, $S(i-2, j)$, $S(i-1, j-1)$,
- for $S(i,j-1)$: $S(i-1, j-2)$, $S(i-1, j-1)$, $S(i, j-2)$

Thus we can write a general recursive definition of all optimal alignment scores $S(i, j)$:

$$S(i,j) = \max \{ S(i-1,j-1) + \sigma(x_i,y_j) ,$$
$$S(i-1,j) + \gamma ,$$
$$S(i,j-1) + \gamma \}$$

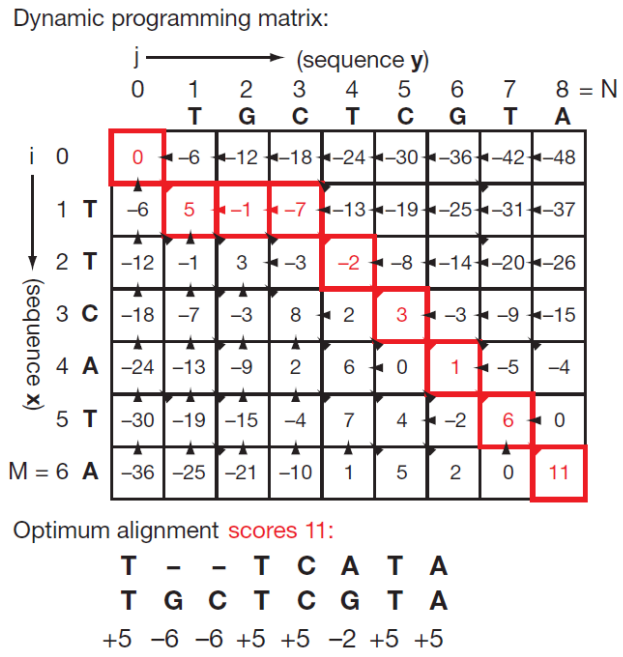


Figure 1 The filled dynamic programming matrix for two DNA sequences, x = TTCATA and y = TGCTCGTA, for a scoring system of +5 for a match, –2 for a mismatch and –6 for each insertion or deletion. The cells in the optimum path are shown in red. Arrowheads are 'traceback pointers,' indicating which of the three cases were optimal for reaching each cell. (Some cells can be reached by two or three different optimal paths of equal score: whenever two or more cases are equally optimal, dynamic programming implementations usually choose one case arbitrarily. In this example, though, the

optimal path is unique.)

2.2.2 Dynamic Programming (DP) matrix

The problem with a purely recursive alignment algorithm may already be obvious. If you look carefully at the nine smaller sub-problems, we can solve in the second round of the top-down recursion. Some sub-problems are already occurring more than once, and this wastage gets exponentially wore as we move deeper into the recursion.

Clearly, the sensible thing to do is to somehow keep track of which sub-problems we are already working on. This is the key difference between DP and simple recursion: a DP algorithm memorizes the solutions of optimal sub-problems in a tabular form (DP matrix), so that each sub-problem is solved just once.

For the pairwise sequence alignment algorithm, the optimal scores S(i,j) are written in a two-dimensional matrix as shown in Fig. 1.

2.2.3 Bottom-up calculation to get the optimal score

Once the DP matrix S(i, j) is given, it is easy to fill it in a bottom-up way, from smallest problems to progressively bigger problems. We know the boundary conditions in the left most column and top most row:

- $S(0, 0) = 0$
- $S(i, 0) = \gamma * i$ (i times of gap penalty)
- $S(0, i) = \gamma * i$

If we initialize the top row and left column with the above values, we can fill in the rest of the matrix by using the previous recursive definition of S(i,j). We calculate any cell where we already know the three values:

- Upper-left (i-1, j-1)
- upper (i-1, j)
- left (i, j-1)

Now, we can fill all the cells in the matrix with the iteration i = 1,..,M and j = 1,…N.

2.2.4 Traceback to get the optimal alignment

Once we have done filling the matrix, the score of the optimal alignment of the complete sequences is the last score we calculate, S(M, N). We still don't know the optimal alignment itself, though. Thus we recover by a recursive 'traceback' of the matrix.

We start in cell (M, N) to

- determine which of the three cases we used to get here(e.g. by repeating the same three

calculations),

- record that choice as part of the alignment,
- and then follow the appropriate path for that case back into the previous cell on the optimum path.

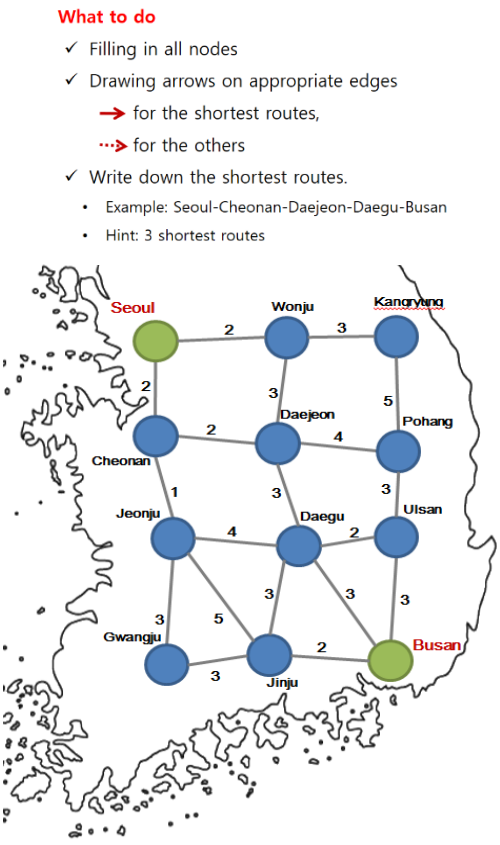
We keep doing this until we reach cell (0, 0) at which the optimal alignment is fully reconstructed. DP is guaranteed to give a mathematically optimal solution. But DP is computational demanding algorithm. This is why there is so much research devoted to fast approximations to DP alignment, like BLAST and FASTA.

3. Prelab activities

3.1 Dynamic programming – finding the shortest path

We want to find the shortest time path from Seoul to Busan on the right figure. The numbers on lines represents the time required between two cities. Finding the shortest path directly is pretty complex. There are some algorithms for finding shortest path between two points, for example, Dijkstra's algorithm, Fibonacci heap, and so on.

Dynamic programming (DP) makes it easy. So, we apply DP to find the path. We can find the shortest path with the following steps.

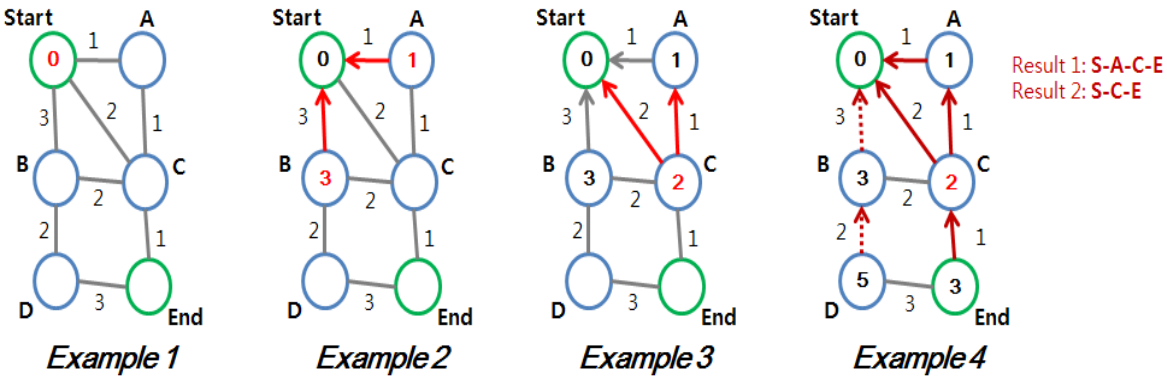


i. Fill the start node with 0. (Example 1) The number in a node represents the shortest time required to get to the start node.

ii. Fill in a neighboring node, and draw an arrow from the node to the previous node resulting in the shortest time. (Example 2) When the scores are tie, the arrows can be more than one. (Example 3)

iii. Iterate step ii, until the filling the end node.

iv. From the end node, back track to obtain the shortest path. The resulting paths can be more than one, when the scores are tie. (Example 4)



3.2 Practice of C++ programming

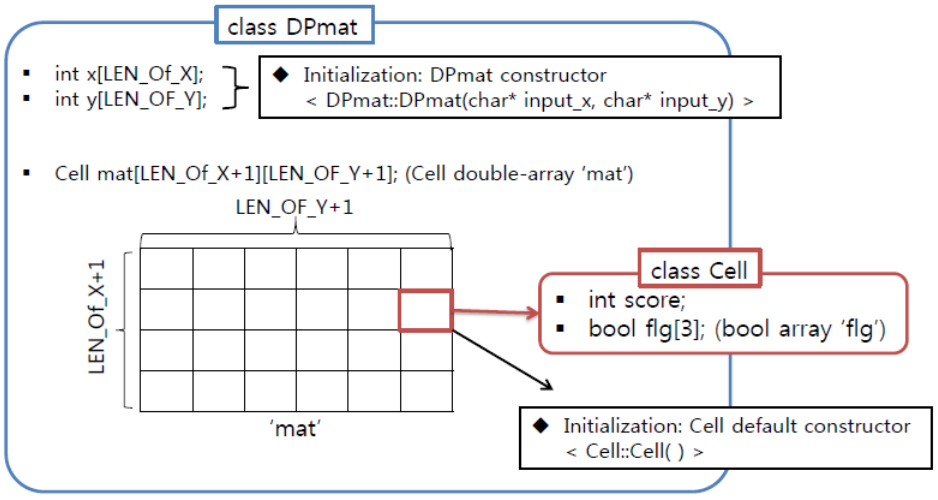
You should practice the C++ programming to apply the dynamic programming. We prepared some C++ files. In the prelab activity, you learn the C++ programming simply and the use of that in the experiment.

There are 5 files prepared: DP_Main.cpp, DPmat.h, DPmat.cpp, Cell.h, and Cell.cpp.
In the experiment, two classes are going to be used. (DPmat, Cell)
Each class has its own class declaration and definition as below.

	DPmat	Cell
Declaration	DPmat.h	Cell.h
Definition	DPmat.cpp	Cell.cpp

It also has one main cpp file. (DP_Main.cpp) You should be well-informed of those 5 files because the class member variables and functions included those files are also used in the mainlab activities.

The following figure shows the structure of two classes.



So, you should program two functions of DPmat.cpp to have the result as the expected results. You don't have to change the other codes.

Function 1 (constructor): DPmat::DPmat(char* input_x, char* input_y)

→ Initialize int* x, int* y, Cell** mat (representing sequence X, sequence Y, and DP matrix, respectively)

Function 2: void DPmat::fill_mat()

→ Fill in Cell** mat with the numbers between 1 and 63, as a prototype version of the mainlab function which fills in DP matrix. Here, choose the proper order based on the function print_mat()

Refer the comments in the code DPmat.cpp. After correcting the DPmat.cpp, compile DP_main.cpp with your favorite tool and compare the results with the expected results. And submit DPmat.cpp file ONLY. The expected result is as following:

```
<< the result of print_xy_int() >>
x: 224121
y: 23424321
<< the result of print_mat() >>
63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37
 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 1
0 9 8 7 6 5 4 3 2 1
```

For the beginners, here are some references for c++ programming.
Absolute c++ 3rd edition by Walter Savitch (book), chapter 6.2 classes ~ chapter 7.1 constructors (p240 ~ p284)

To modify and compile the C++ file, please use **Codeblock** for window or **g++ compiler** in linux or mac.

4. Mainlab activities

- Sequence alignment experiment: our objective is to confirm the aligned sequence given two arbitrary input sequences with dynamic programming algorithm
- There are three primary steps to achieve above: Drawing DP Matrix, Back-Tracking, Handling multiple results

4.1 Drawing DP Matrix

① We have two DNA sequences x=TGCTCA and y=TGCTCGTA and want to develop a program for aligning the shorter sequence x to the longer one y. We have a score systems $\sigma(x_i, y_j)$: You must use 'scoring_mat' defined in DPmat.cpp for match & mismatch cases.

② Complete the function void DPmat::fill_in_DPmat() in DPmat.cpp. You can modify existing functions and add new functions if needed. Again, scoring matrix is already implemented in DPmat.cpp.

③ Initialize the first row and the first column of DPmat and fill in the remaining cells of DPmat with a score S(i, j) and a direction flg(i, j). Perform following steps to get a score and direction.

Step 1. Calculate three scores

Step 2. Compare scores and choose the highest one.

Step 3. Save the highest one as score S(i, j) with its direction

Demo Question 1. Check out; The expected result is following:

```
<< score & flag of matrix>>
[0 0 0 0]    [1 0 0 -6]    [1 0 0 -12]    [1 0 0 -18]    [1 0 0 -24]    [1 0 0 -30]    [1 0 0 -36]    [1 0 0 -42]    [1 0 0 -48]
[0 0 1 -6]   [0 1 0 5]     [1 0 0 -1]     [1 0 0 -7]     [1 1 0 -13]    [1 0 0 -19]    [1 0 0 -25]    [1 1 0 -31]    [1 0 0 -37]
[0 0 1 -12]  [0 0 1 -1]    [0 1 0 10]     [1 0 0 4]      [1 0 0 -2]     [1 0 0 -8]     [1 1 0 -14]    [1 0 0 -20]    [1 0 0 -26]
[0 0 1 -18]  [0 0 1 -7]    [0 0 1 4]      [0 1 0 15]     [1 0 0 9]      [1 1 0 3]      [1 0 0 -3]     [1 0 0 -9]     [1 0 0 -15]
[0 0 1 -24]  [0 1 1 -13]   [0 0 1 -2]     [0 0 1 9]      [0 1 0 20]     [1 0 0 14]     [1 0 0 8]      [1 1 0 2]      [1 0 0 -4]
[0 0 1 -30]  [0 0 1 -19]   [0 0 1 -8]     [0 1 1 3]      [0 0 1 14]     [0 1 0 25]     [1 0 0 19]     [1 0 0 13]     [1 0 0 7]
[0 0 1 -36]  [0 0 1 -25]   [0 0 1 -14]    [0 0 1 -3]     [0 0 1 8]      [0 0 1 19]     [0 1 0 23]     [0 1 0 18]     [0 1 0 18]
```

Final Report Question 1.

Describe differences between Needleman-Wunsch algorithm (Global alignment) and Smith-Waterman algorithm (Local alignment).

4.2 Back-tracking

① Complete the function void DPmat::trace_back() in DPmat.cpp. Also, you can modify existing functions and add new functions if needed.

② In DPmat::trace_back() function, trace back the direction from the end point of the matrix, mat[length of sequence X][length of sequence Y], by recursive process.

Demo Question 2. Check out; The expected result is following:

```
<< score & flag of matrix>>
[0 0 0 0]    [1 0 0 -6]    [1 0 0 -12]    [1 0 0 -18]    [1 0 0 -24]    [1 0 0 -30]    [1 0 0 -36]    [1 0 0 -42]    [1 0 0 -48]
[0 0 1 -6]   [0 1 0 5]     [1 0 0 -1]     [1 0 0 -7]     [1 1 0 -13]    [1 0 0 -19]    [1 0 0 -25]    [1 1 0 -31]    [1 0 0 -37]
[0 0 1 -12]  [0 0 1 -1]    [0 1 0 10]     [1 0 0 4]      [1 0 0 -2]     [1 0 0 -8]     [1 1 0 -14]    [1 0 0 -20]    [1 0 0 -26]
[0 0 1 -18]  [0 0 1 -7]    [0 0 1 4]      [0 1 0 15]     [1 0 0 9]      [1 1 0 3]      [1 0 0 -3]     [1 0 0 -9]     [1 0 0 -15]
[0 0 1 -24]  [0 1 1 -13]   [0 0 1 -2]     [0 0 1 9]      [0 1 0 20]     [1 0 0 14]     [1 0 0 8]      [1 1 0 2]      [1 0 0 -4]
[0 0 1 -30]  [0 0 1 -19]   [0 0 1 -8]     [0 1 1 3]      [0 0 1 14]     [0 1 0 25]     [1 0 0 19]     [1 0 0 13]     [1 0 0 7]
[0 0 1 -36]  [0 0 1 -25]   [0 0 1 -14]    [0 0 1 -3]     [0 0 1 8]      [0 0 1 19]     [0 1 0 23]     [0 1 0 18]     [0 1 0 18]

y: TGCTCGTA
x: TGCTCA

z: TGCTC--A     score: 18
```

Demo Question 3. Give brief explanation of your implementation (why it works).

4.3 Handling multiple results

① Now we will run your program with different input sequence, X=TTCCG and Y=TAACTCG. Your program should works without any modification of code if you programmed right way. The result should include DP matrix with flag and score, and aligned sequences Z_1, Z_2 ... with their final score.

Demo Question 4. Check out; The expected result is following:

```
<< score & flag of matrix>>
[0 0 0 0]      [1 0 0 -6]     [1 0 0 -12]    [1 0 0 -18]    [1 0 0 -24]    [1 0 0 -30]    [1 0 0 -36]    [1 0 0 -42]
[0 0 1 -6]     [0 1 0 5]      [1 0 0 -1]     [1 0 0 -7]     [1 0 0 -13]    [1 1 0 -19]    [1 0 0 -25]    [1 0 0 -31]
[0 0 1 -12]    [0 1 1 -1]     [0 1 0 4]      [1 1 0 -2]     [1 0 0 -8]     [0 1 0 -8]     [1 0 0 -14]    [1 0 0 -20]
[0 0 1 -18]    [0 0 1 -7]     [0 1 1 -2]     [0 1 0 3]      [0 1 0 3]      [1 0 0 -3]     [0 1 0 -3]     [1 0 0 -9]
[0 0 1 -24]    [0 0 1 -13]    [0 1 1 -8]     [0 1 1 -3]     [0 1 0 8]      [1 0 0 2]      [0 1 0 2]      [1 0 0 -4]
[0 0 1 -30]    [0 0 1 -19]    [0 0 1 -14]    [0 0 1 -9]     [0 0 1 2]      [0 1 0 6]      [1 0 0 0]      [0 1 0 7]

y: TAACTCG
x: TTCCG

z: TT-C-CG     score: 7
z: T-TC-CG     score: 7
```

Final Report Question 2.

Describe BLOSUM matrix and its use.

Final Report Question 3.

Result from sequence alignment can be used as measure of similarity between two sequences. Using your sequence alignment program, find father's true and only offspring. Your screenshot which justify your similarity measure should be included in the submission.

Father	ATGACCGTAATAGGT
Candidate 1	AGTGGTAACT (0)
Candidate 2	TGACAGTACT (8)
Candidate 3	AACCTTGTCT (1)

5. Reference

[1] Sean R. Eddy, "What is dynamic programming", Vol. 22, No. 7, July 2004, Nature Biotechnology

[2] Ingvar Eidhammer, Inge Jonassen, and William R. Taylor, Protein Bioinformatics an algorithmic approach to sequence and structure analysis, 2004, WILEY.