Programming assignment for the course
“Sequence Analysis” (2006)

Original text by John W. Romein,
adapted by Bart van Houte (bart@cs.vu.nl)

Introduction

Please note: This assignment is only obligatory for students doing the Bioinformatics Master. Other
students who know how to program are also strongly encouraged to do this assignment, however.

In this assignment, you will have to implement some of the most basic alignment algorithms in bioin-
formatics. There will be no fixed hours for this practical course, and you can work independently in
one of the computer rooms that is available. Simple questions can be directed by email to your advisor:
bart@cs.vu.nl. You can also ask for oral assistance: room P1.38.

The programming is done on the Solaris systems of the Informatics department (of course, you program
in a clean and portable way, so it actually does not matter on which platform you program). You can
program in either Java or C. For both languages, we provide templates that give some basic support for the
programs you are to implement. The templates, which can be found on the website of the course, contain
functions that read the sequences from the command line and also contain some default exchange matrices.
To facilitate testing, programs have to adhere strictly to the input and output formats described below.

Language documentation

Information on the Java programming language can be found online via

• http://java.sun.com/docs/index.html (general information)
• http://developer.java.sun.com/developer/codesamples/basics.html (simple examples)
• http://developer.java.sun.com/developer/codesamples/ (more examples; go to “Essen-
tial APIs”)  
• http://java.sun.com/j2se/1.4.2/docs/api/ (the API (Application Programming Interface))

The library also has a couple of books available.

Program submission

Submit each program by email to your advisor. Each program should compile flawlessly with “cc align.c
-o align” (C) or “javac Align.java” (Java). The programs must also handle the example tests given
below. Turn in your program no later than:

<table>
<thead>
<tr>
<th>exercise</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>November 17th</td>
</tr>
<tr>
<td>1b</td>
<td>November 24th</td>
</tr>
<tr>
<td>2</td>
<td>December 15th</td>
</tr>
</tbody>
</table>
Each program is graded as follows:

- Functionality (40%) — each program is tested and graded with respect to the test results.
- Programming style (60%) — each program is graded with respect to the readability and cleanliness of the source code. This includes documentation: briefly mention what nontrivial functions do and how they do it (as comments in the source code), but do not overdo it (for example, a statement like “x = 1;” needs no further explanation). The template gives an impression of the right amount of documentation.

**Exercise 1: global alignments**

For this exercise, you are required to implement the Needleman-Wunsch global alignment algorithm [1] with a simplified gap-penalty scheme, as explained during the theory part of the course. The algorithm consists of two phases: during the first phase, the dynamic-programming score matrix is computed, and during the second phase, a traceback is performed to find the optimal alignment. For exercise 1a, you have to implement the part that computes the score matrix, and for exercise 1b, you have to implement the traceback routine.

**Exercise 1a**

Extend the template with a function that computes the dynamic-programming score matrix. You must also add an option “-v” that prints the score matrix (see below).

Matrix entries should be computed according to Equation 1:

\[
M_{0,0} = 0 \\
M_{x,y} = \max \begin{cases} 
M_{x-1,y-1} + E_{S_{n-1}, S_{n-1}} & \text{if } x \geq 1 \land y \geq 1 \\
M_{x-1,y} - g & \text{if } x \geq 1 \\
M_{x,y-1} - g & \text{if } y \geq 1
\end{cases}
\] (1)

where \( M \) is the score matrix; \( E \) is the exchange matrix (the exchange matrix is given in the template file, so you need not write this yourself); \( S_0 \) and \( S_1 \) are the two sequences where \( S_n \) is the first residue in the \( n \)-th sequence; and \( g \) is the gap penalty. For this exercise, you need not implement an affine gap penalty; you can simply apply a penalty of \( g \) for each residue that matches a gap.

The template contains a function that reads two sequences from an input file; the file should be in FASTA format. In this format, each sequence starts with one line beginning with a ‘>’ symbol, immediately followed by the name and/or a description of the sequence, extending to the end of the line. The sequence residues themselves appear on subsequent lines, up to the point where a new sequence name starts (or the end of the file). The program should align the first two sequences appearing in the input file; the name of the file is given a command line argument. An example file test.fasta is in the template directory and contains the following sequences:

```
>seq1
YICSFADCCF
>seq2
FPCKECECA
```

The code in the template already reads in the two sequences (or terminates with an error message if the input does not conform to FASTA). The template also provides an option to select one of the built-in exchange matrices: valid options are “-e pam250”, “-e blosum62”, and “-e identity”. The default is pam250. The gap penalty can be changed with the “-p” option; the default is 2.
Using the default PAM250 matrix and default gap penalty 2, the command “align -v test.fasta” (C) or “java Align -v test.fasta” (Java) should print the following score matrix:

```
- 0  F  P  C  K  E  E  C  A
Y -2 7 5 3 1 -1 -3 -5 -7
I -4 5 5 3 1 -1 -3 -5 -6
C -6 3 3 17 15 13 11 9 7
S -8 1 4 15 17 15 13 11 10
F -10 1 2 13 15 13 11 9 8
A -12 -1 2 11 13 15 13 11 11
D -14 -3 0 9 11 16 18 16 14
C -16 -5 -2 12 10 14 16 30 28
C -18 -7 -4 10 8 12 14 28 28
F -20 -9 -6 8 6 10 12 26 26
```

The first sequence is printed vertically, and the second sequence horizontally. Note that the width of each printed number is 5 characters, and that the first row and first column show the residues in the sequences.

Tip 1: proceed as follows:

- Allocate memory for the dynamic-programming matrix.
- Add code to recognize the option “-v”.
- Write a function that prints the matrix if the user supplies “-v” on the command line (even though the matrix values have not been computed yet). If you program in Java, you will find that there is no standard function that prints an integer over a five-column-width field. The template contains a method “printFormatted()” that can be used for this purpose.
- Write a function that computes the entries in the dynamic-programming matrix. Since each matrix entry depends on the entries left, above, and above-left, the easiest way is to compute all matrix entries row-wise from top to bottom, and each row from left to right.

Tip 2: test your program after each of these steps!
Tip 3: also test your program with very short (1 residue) and long sequences, and verify whether the output is what you expect.

**Exercise 1b**

For this exercise, you are required to extend the program of exercise 1a with a traceback function that yields the optimal alignment. Eventually, the program must print the alignment as follows. On the first line: the first sequence, possibly with gaps. On the second line: a pipe symbol (‘|’) if two vertically matched residues are equal to each other (i.e., the character on the line above a position is equal to the character on the line below the position); a space otherwise. On the third line: the second sequence, possibly with gaps. On the fourth line: the score, in the form score = n. You need not break up long alignments that do not fit an eighty-column screen.

For example, invoking the program “align test.fasta” (C) or “java Align test.fasta” (Java) should print the following output:

```
YICSFADCCF
| | |
FPCK--EECA-
score = 26
```
Tip: proceed as follows:

- Allocate two empty strings that will eventually hold the aligned sequences (with gaps). If you program in Java, use the “java.lang.StringBuffer” class (see the API) to construct the aligned sequences. If you program in C, reserve enough memory for the aligned sequences; since gaps may be introduced, the sequences may be longer than the original sequences.

- Construct the aligned sequences from the right to the left. Start from the bottom-right matrix entry. Check where we came from: you either came from left, above, or from above-left (see Equation 1). Test whether the current entry equals the above-left entry plus the exchange value. If so, we apparently matched two residues, thus prepend one residue to the one sequence and the other residue to the other sequence. If not, check whether the current entry equals the entry above it minus the gap penalty. If so, we insert a gap; prepend a gap symbol “’-’” to the one sequence and the corresponding residue to the other sequence. Otherwise, prepend a gap symbol to the other sequence and the corresponding residue to the one sequence. Repeat this process until you are in the upper-left corner of the matrix.

- Print both aligned sequences and the score, after which you can write code that prints a line with vertical bars between equal residues.

- Try your program with different sequences as well, including those that yield gaps at the beginning and/or ending of one of the sequences.
Exercise 2: semi-global and local alignments

For this exercise, you have to implement the semi-global alignment algorithm and the Smith-Waterman local alignment algorithm [2], as explained during the lectures. You should extend the program you wrote for Exercise 1, so the program should still support global alignments as well. The program should perform a semi-global alignment if “-s” is supplied as command-line argument, or a local alignment if “-l” is given (the code to recognize the arguments is already there in the template). If neither “-s” nor “-l” is given, the program should perform a global alignment. Independent of the type of alignment (global, semi-global, or local), the program should print the score matrix if “-v” is given, using the same format as in Exercise 1a. In each case, it should print the alignment, also in the same format as in Exercise 1b.

Matrix entries for semi-global alignments should be computed according to Equation 2:

\[
M_{x,y} = \max \begin{cases} 
0 & \text{if } x = 0 \lor y = 0 \\
M_{x-1,y-1} + E_{S_{x-1},S_{y-1}} & \text{if } x \geq 1 \land y \geq 1 \\
M_{x-1,y} - g & \text{if } x \geq 1 \\
M_{x,y-1} - g & \text{if } y \geq 1 \\
\end{cases}
\]

and matrix entries for local alignments should be computed according to Equation 3:

\[
M_{x,y} = \max \begin{cases} 
M_{x-1,y-1} + E_{S_{x-1},S_{y-1}} & \text{if } x \geq 1 \land y \geq 1 \\
M_{x-1,y} - g & \text{if } x \geq 1 \\
M_{x,y-1} - g & \text{if } y \geq 1 \\
0 & \text{otherwise}
\end{cases}
\]

where the variables are the same as in Exercise 1.

The example test file from Exercise 1 should print the following semi-global alignment:

```
- F P C K E E C A
  0 0 0 0 0 0 0 0 0
  Y 0 7 5 3 1 -1 -2 -2
  I 0 5 5 3 1 -1 -3 -2 -1
  C 0 3 3 17 15 13 11 9 7
  S 0 1 4 15 17 15 13 11 10
  F 0 9 7 13 15 13 11 9 8
  A 0 7 10 11 13 15 13 11 11
  D 0 5 8 9 11 16 18 16 14
  C 0 3 6 20 18 16 16 30 28
  C 0 1 4 18 16 14 14 28 28
  F 0 9 7 16 14 12 12 26 26

YICSFADCCF
|   |
FPCK--EECA--
score = 28
```

and perform the traceback from the highest-scoring entry in the last column or bottom row (if multiple ones exist, it is preferable to start from the one that is the closest to the lower righthand-side corner, since it leads to shorter alignments. For example, starting from the rightmost entry on the third last row in the example above yields

```
YICSFADC--CF
|   |
FPCK--EECA--
score = 28
```

which is longer; it is better to start from the highest-scoring entry on the last row or bottom column that is most close the lower righthand-side corner).
The scores for the local alignment looks as follows:

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>F</th>
<th>P</th>
<th>C</th>
<th>K</th>
<th>E</th>
<th>E</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>17</td>
<td>15</td>
<td>11</td>
<td>12</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>15</td>
<td>17</td>
<td>15</td>
<td>11</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>9</td>
<td>7</td>
<td>13</td>
<td>15</td>
<td>11</td>
<td>9</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>15</td>
<td>13</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>16</td>
<td>16</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>20</td>
<td>18</td>
<td>16</td>
<td>30</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>18</td>
<td>16</td>
<td>14</td>
<td>28</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>9</td>
<td>7</td>
<td>16</td>
<td>14</td>
<td>12</td>
<td>26</td>
<td>26</td>
<td></td>
</tr>
</tbody>
</table>

YICSFADC  
|   |   |
| FPCK-EEC  |

score = 30

The traceback must start from the highest-scoring entry that can be anywhere in the matrix, and can stop as soon as a zero is found (thus not necessarily in the upper lefthand-side corner).

If multiple highest-scoring alignments exist, you can print an arbitrary one. It is also allowed to print them all, but this harder since it requires recursion to perform backtracking.

Note that it is not strictly necessary to stick to the structure of the template program. For example, you are allowed to write one method that can perform either type of alignment, but you may also write separate methods that can perform a single type of alignment. The same holds for the traceback.

Before submitting your program, do not forget to test it with different input sequences, including ones that match poorly on the starting and/or ending sides of the sequences, so that the resulting local alignments become short. Also, two sequences that mismatch over the entire sequence length, should yield a semi-global alignment where the sequences are “drifted apart”: the resulting alignment contains many leading gaps in one sequence and many trailing gaps in the other sequence, such that (nearly) all residues match a gap.

References
