Assignment 4 — Dynamic Programming

Many problems in bioinformatics amount to approximate string matching. For example, whenever a new gene is sequenced, searching known genes for close matches allows us to attempt to infer the new gene's functions.

1 Sequence Alignment

Specifically, a gene is a string over the alphabet $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$, and the closeness of two genes is measured by the extent to which they are aligned. An ***alignment*** of two strings, x and y, is an arrangement of their characters in columns, in the same order as they originally appeared, potentially interspersed by spaces.

For example, given $x = \texttt{ACGAT}$ and $y = \texttt{TACGCA}$, one possible alignment is:

```
A  -  -  -  C  G  A  T
T  A  C  G  C  -  -  A
```

Where the '–' character is used to indicate a space. Note that a typical further requirement (and will be a requirement for this assignment) is that no column may contain two spaces.

Another possible alignment is:

```
-  A  C  G  -  A  T
T  A  C  G  C  A  -
```

Of these two alignments, the second appears to be "better", but how can we quantify "better"? The ***score*** of an alignment is specified by a $(|\Sigma| + 1) \times (|\Sigma| + 1)$ ***scoring matrix***, δ. For example, the latter of the above alignments has a score of $\delta(\texttt{-}, \texttt{T}) + \delta(\texttt{A}, \texttt{A}) + \delta(\texttt{C}, \texttt{C}) + \delta(\texttt{G}, \texttt{G}) + \delta(\texttt{-}, \texttt{C}) + \delta(\texttt{A}, \texttt{A}) + \delta(\texttt{T}, \texttt{-})$.

In your programming language of choice, implement a dynamic programming algorithm to find the highest scoring alignment of two strings.

You may assume that both of the given strings will be non-empty and will contain only the characters '`A`', '`C`', '`G`', or '`T`'. You may also assume that the scoring matrix will always contain exactly 5 rows and 5 columns, each in the order '`A`', '`C`', '`G`', '`T`', '`-`', will always be symmetric about the major diagonal, and will always provide integer scores.

For example, the above strings, along with a simple scoring matrix, would be represented as:

```
ACGAT
TACGCA
x A C G T -
A 1 0 0 0 0
C 0 1 0 0 0
G 0 0 1 0 0
T 0 0 0 1 0
- 0 0 0 0 0
```

Since this scoring matrix only rewards exact matches, in this situation, the second of the above alignments, with a score of 4, would indeed be better than the first, with a score of 1 (and, in fact, is the highest scoring alignment overall).

Your program must accept as a command line argument the name of a file containing two strings and a scoring matrix as described above, then print to `stdout` the highest scoring alignment as follows.

- The first given string, named x, must be printed above the second given string, named y
- The columns of the alignment must be space-separated, and '–' characters must be used to represent spaces within the aligned strings.
- After printed the alignment, print the score.

For example:

```
$ ./compile.sh
$ ./run.sh test_files/in1.txt
x: - A C G - A T
y: T A C G C A -
Score: 4
```

You may first assume that the highest scoring alignment will be unique. Your program will be tested using `diff`, as its printed output must match *exactly*.

2 Submission

The following items must be demonstrated/presented to me in lab on the day of the deadline.

- Definition, base cases, formula, and solution for a dynamic programming algorithm to find the highest scoring alignment of two strings.

The following files are required and must be pushed to your GitHub Classroom repository by the deadline:

- `compile.sh` — A bash script to compile your submission (even if it does nothing).
- `run.sh` — A bash script to run your submission.
- `*.py` or `*.java` — Your source code in Python or Java of a working program.