Project 1: Basic exact pattern matching

This project is about exact pattern matching. You should implement the naive quadratic time algorithm and either the border-array or the KMP-algorithm. The naive algorithm has the worst case running time O(nm) and the other algorithms have worst-case running time O(n+m).

The algorithms should be implemented in two programs, `search-naive` and `search-ba` or `search-kmp`. Both programs should take two arguments, the first should be a FASTA file and the second a FASTQ file. It should output all matches in the SAM format to stdout (i.e., it should write to the terminal). Since we are only doing exact matching, the CIGAR strings in the output should consist of M's only, since all characters in the read will match the reference at the reported position.

For example, with this FASTA file

```
>one
mississippi
>two
mississippimississippi
```

and this FASTQ file

```
@iss
iss
+
~~~
@mis
mis
+
~~~
@ssi
ssi
+
~~~
@ssippi
ssippi
+
~~~~~~
@nope
nope
~~~~
```

your output should be

```
iss 0   one 2   0   3M  *   0   0   iss ~~~
iss 0   one 5   0   3M  *   0   0   iss ~~~
iss 0   two 2   0   3M  *   0   0   iss ~~~
iss 0   two 5   0   3M  *   0   0   iss ~~~
iss 0   two 13  0   3M  *   0   0   iss ~~~
iss 0   two 16  0   3M  *   0   0   iss ~~~
mis 0   one 1   0   3M  *   0   0   mis ~~~
mis 0   two 1   0   3M  *   0   0   mis ~~~
```

```
mis 0   two 12  0   3M  *   0   0   mis ~~~
ssi 0   one 3   0   3M  *   0   0   ssi ~~~
ssi 0   one 6   0   3M  *   0   0   ssi ~~~
ssi 0   two 3   0   3M  *   0   0   ssi ~~~
ssi 0   two 6   0   3M  *   0   0   ssi ~~~
ssi 0   two 14  0   3M  *   0   0   ssi ~~~
ssi 0   two 17  0   3M  *   0   0   ssi ~~~
ssippi  0   one 6   0   6M  *   0   0   ssippi  ~~~~~~
ssippi  0   two 6   0   6M  *   0   0   ssippi  ~~~~~~
ssippi  0   two 17  0   6M  *   0   0   ssippi  ~~~~~~
```
assuming you iterate over reads in an outer loop and FASTA records in an inner loop. If you order your loops differently, of course the output will be different.

The project should be in groups of 2–3 students. It will not be graded.

Part 1: parsers

Write parsers for FASTA and FASTQ and programs that take a file as input, parses, and output the result. Compare your output files with your input files. These should be identical.

For testing the running time as functions of n and m, you should also write code for generating FASTA and FASTQ files (with appropriate properties for your tests).

Part 2: mappers

Now write the tools for exact pattern matching. You can use the naive algorithm to test your linear time algorithm; the result of the two programs that you write should be identical.

Evaluation

You should upload your solution to Blackboard before the lecture **19/2**.

The solution you upload should contain code for building two programs, `search-naive` and `search-kmp` (with the appropriate build tools for the programming language you have chosen), and a one-page PDF file where you address the following questions:

Insights you may have had while implementing and comparing the algorithms. * Problems encountered, if any. * An experiment that verifies the correctness of your implementations. * An experiment that verifies that your implementation of `search-naive` uses no more time than O(nm) to find all occurrences of a given pattern in a text. Remember to explain your choice of test data. What are “best” and “worst” case inputs? * An experiment that verifies that your implementations of `search-ba` or `search-kmp` use no more time than O(n+m) to find all occurrences of a given pattern in a text. Remember to explain your choice of test data. What are “best” and “worst” case inputs?