CSC 211—Fundamental Data Structures
Compare the brute-force (BF), Rabin-Karp (RK) and Boyer-Moore (BM) pattern-matching methods.

Practical 6 Term 2 23 May 2016

Submit today as soon as possible before leaving the practical.

Do not decorate your program with nonsensical or obvious descriptions of your code, only insert useful comments that will remind you or inform others of what your code does. Comments can be a waste of time, so use them sparingly. If your code changes ensure that your comments are updated to reflect the changes. Put the ⟨name of the *file*⟩ in a '//' comment on the first line of each of your files AND your ⟨Surname, Firstname and Student number⟩ *must* also be given in the *second* line of every file in a '//' comment. Use meaningful variable names to enhance code readability.

Copy a previous working practical into today's `62practical` directory and clean it up. Then implement the brute-force (BF) pattern-matching method and the Rabin-Karp (RK) pattern-matching method and the the Boyer-Moore (BM) pattern matching method. Your main method must be in a class called `tryMatch`. Use the large set of words in the file

`/export/home/notes/ds/lesmiserables.text`—this might be changed to `ulysses.text`.

in the SunLab as the file in which you must look up and count appearances of these words that we call patterns. Do not copy this file to your working directory—penalty 20% off this practical after marking. A supply of such patterns to be used is stored in the file `patterns.text` in the same directory—do not copy this file to your working directory—penalty another 20% off this practical after marking.

First the entire text file must be read into the array `String T` and then the patterns must be read into your program and then processed *one-by-one* as the variable `String p`. Each pattern must be looked up using one of each of the methods. For each method the program must run for about 1 second and then report how many patterns were found in the time of one second. Your pattern matching routines must search repeatedly for a pattern—a word in lowercase—until all its occurrences have been found and report on the number of its appearances. Write this to a file.

The final output of your code to the screen is a table that gives the name of each method and the number of patterns whose first occurrence it has matched within 1 second.

We advise you to test your code by checking if it works for the first 10 (ten) patterns. When this works, proceed to do the timing. Do all the timing for one method before proceeding to the next one. If you read any uninteded ambiguity into this problem statement please resolve it yourself.

1. *Preparation of file system*: Today's practical must be done as usual inside your name file that lies in your home directory. Do today's work in a directory called `62practical`. Create the new directory by copying and old working directory into the new one. Fix up the `Makefile` so that it processes today's work by invoking *make* on the command line interface (CLI).

2. **Today**: First create a brute force (BF) pattern searching method that works for a fixed piece of text called `T` and a smallish but fixed pattern called `p`.

3. **Today**: Program your own method for pattern searching with the Rabin-Karp (RK) method, test it, and when the method is shown to work, run the timed experiment.

4. **Today**: Create a pattern searching method for the Boyer-Moore (BM) method, test it, and when the method is shown to work, run the timed experiment.

   Your coding skills will improve if the code is always your own. Do your debugging by first getting each method to work with a small text `T` and pattern `p` that are assigned in the test code before trying to apply the search methods on the larger text and patterns from the files. Also do tests that (1) succeed at the start of `T`, (2) somewhere away from the end points of `T`, and (3) one that matches at the end of `T`, for example:

```
String p = "The";
String T = "The Rabin-Karp method is much faster that the brute-force method.";

String p = "brute-force";
String T = "The Rabin-Karp ethod is much faster that the brute-force method.";

String p = "method.";
String T = "The  Rabin-Karp method is much faster that the brute-force method.";

String p = "";
String T = "The  Rabin-Karp method is much faster that the brute-force method.";

String p = "method.";
String T = "";

String p = "";
String T = "";
```

   Please note that these examples are textual, but the actual practical uses strings that consist entirely of strings separated by spaces or newline characters. Once you are convinced that your code works for these examples the program must be enhanced to read a pattern `p` from the *patterns.text* file and search for that pattern once in the text `T` that has also been read from a file. *Do not copy the data.* Simply read it and use it to generate your results. The data is stored in the files given above.

   - Time the BF method for 1 second and report the number n of patterns that were searched.
   - Do the same for the RK method.
   - Do the same for the BM method.

5. Submit ALL your pracs in a file that has been created in the usual manner: i.e., `cd; make submit`.

6. **The pattern drawing strategy.** Draw the patterns from the `patterns.text` file. For each $i \in [1..P]$, draw a pattern p_i from the `patterns.text` file and then search through the text for its occurrence in the `lesmiserables.text` file. Note that some patterns might not be present in the text file.

7. Note that all the occurrences of a pattern must be looked up and counted.

8. In summary look up each "pattern" p throughout the entire "text" file T.