CS337 Project 5
String Matching Algorithms

TA in charge: Lara Schmidt

Due Friday, 3rd May 2013, 11:59PM

1 Note

This project is voluntary. It will give 5% (if you get 100%) to your grade, applied after the curve (if there is one). It will be graded a little more harshly than previous projects.

2 Overview

The purpose of this project is to learn about string matching by comparing multiple string searching algorithms to see under what circumstances one algorithm may outperform another.

Your job is to implement the Rabin-Karp algorithm and Knuth-Morris-Pratt algorithm, as described in class, to search an input text file for each requested string from a different input patterns file. The program should return an indication if the patterns are found, as well as timing information about the search algorithm. You will be provided with a large text document and a set of pattern strings.

The writeup of the project should explain your results and give a summary of the performance for the different algorithms you have implemented.

Do not use any string matching or search functions that are built into Java when writing your algorithm implementations. Also, you may use whatever you wish when reading in the search strings file, but do not use search or find routines when reading in the source file.

3 Performance

This project is about comparing different algorithms. As such, the program you write must terminate in a reasonable amount of time for all strings provided. If your program does not perform adequately, then there is something wrong with your program and it must be examined. Common inefficiencies may include: failure to properly buffer blocks of characters from the disk drive; using an inefficient data structure; repeating steps unnecessarily; or any number of simple mistakes.

Since the project is partly about algorithm efficiency, it may prove fun to see how fast you can make your algorithms run. To that end, the fastest 3 projects submitted will be

```
&Into the fray good men&
&To be, or not to be&
&There's a blank space at the end of this search string &
&This string has a line-feed
that must be matched&
```

Figure 1: Sample pattern string file. Any valid filename should be accepted.

given 5 additional bonus points (not over 100%). "Fastest" will be defined as the total real runtime of your program from start to finish when running on a dedicated machine. Grading software will be used to measure your projects total execution time so you do not need to implement any additional timing checks. When running the time trials I will be using a different test case than the one provided.

4 User Interface

Your program needs to take in three command-line parameters: the first argument is the name of the pattern file containing the strings to find, second is the name of the text file to search, third is the name of the file to output the match results. Remember, each of these filenames can be arbitrary. The main Java class needs to be named `StrMatch.java`. An example command-line might look like:

```
java StrMatch strings.txt somesource.txt results.txt
```

5 Input

The pattern strings file will be a text file with strings separated by blank lines and/or newlines and/or spaces. Each search string will be surrounded by an ampersand ('&') symbol. See Figure 1 for a sample file. For simplicity you may assume that ampersands ('&') will not appear in the pattern strings themselves.

Any input pattern file may be used with your project. Please use the provided pattern file and input text file for the algorithm comparison in your report. The files have been made available online on Piazza.

5.1 Newlines

Because you are dealing with text files, machine dependent newlines must be detected and allowed to match. Your program needs to be smart enough to know that a multi-line pattern string that contains Windows newlines may match a multi-line text string that contains Unix newlines, and vice versa.

Recall that Windows uses two characters to represent a text newline ($\backslash r \backslash n$) (Hex `0x0D, 0x0A`), while Linux & Mac (Hex `0x0A`) only use one. Your algorithm must have these three characters match so that multi-line pattern strings can be used by your program.

Input File:

```
&To be, or not to be&
&That there's a blue cucumber!&
&Hippocampelephantocamelos&
```

results.txt:

```
RK  FAILED: To be, or not to be
KMP FAILED: To be, or not to be
RK  MATCHED: That there's a blue cucumber!
KMP MATCHED: That there's a blue cucumber!
RK  MATCHED: Hippocampelephantocamelos
KMP MATCHED: Hippocampelephantocamelos
```

Figure 2: Sample pattern file and the corresponding output file `results.txt`. *These are real quotes from real plays. Can you guess the source file?*

In other words, if your pattern string contains a newline represented by `0x0A` it should match a pattern that contains `0x0D 0x0A`; assuming that the rest of the pattern matches as well. The same should be true for matches going the other way, from 2 character to 1 character newlines.

6 Output

See Figure 5 for a sample pattern file and the EXACT format expected in the output file. You may use Standard-Out or create another file to store the statistics needed to study your algorithm implementations. Each search algorithm needs to find the first match and can then terminate the search for that pattern. When running time trials for your report it is important to remember that there are different types of time: CPU time, real time, user time, and so on. It is quite possible that the real time taken to run your algorithm may be much longer than the CPU time taken because of process scheduling on a shared computer, disk I/O and screen output. But then, I know you'll explain all the different things you encounter in your report.

7 Report in `readme.txt`

In `readme.txt` you need to give a report comparing the performance of Rabin-Karp to Knuth-Morris-Pratt when using the pattern strings provided with the sample text file.

When creating your own test cases, be sure to look at various sizes of strings and perhaps even some degenerate cases such as files full of a single letter. You may want to compare your implementations with Java's built-in search routines to see how they compare.

The report for this project will be worth 25% of the project grade.

8 Turning in your project

Please follow the project protocol for files that should be turned in. Put your name, and your partner's name in the report. Students are not getting credit on projects because they

are forgetting to include their names.

Use the Linux turn-in software to submit your project. The command will look like:

```
turnin -submit lschmidt project5 your_files
```

After you have turned in your project it is now possible to use the nifty turnin "`--verify`" flag to check that your files have been turned in properly.

9 Questions and Project Clarifications

Clarifications regarding this project will be posted to Piazza. Though I do not expect any changes, you are responsible for any modifications found there.