**21-04-2017**

**SIMEON BINIATIDIS**

**SolarWinds Coding Test**

**String Search in Text or Binary Files**

I attacked the String Search coding test with several different Java Netbeans projects to divide and conquer the quite complex problem. I didn’t have the luxury of time to put all 3 or 4 different classes together in a unified packaged solution.

**SearchStringInBinaryOrTextFile:**

In this project I’m making heavy use of regular expressions to match the string pattern given, to lines from the binary or text file. I use java class ‘Pattern’ to get the compiled representation of the regex.

A regular expression, specified as a string, must first be compiled into an instance of this class. The resulting pattern can then be used to create a [Matcher](https://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher.html) object that can match arbitrary [character sequences](https://docs.oracle.com/javase/7/docs/api/java/lang/CharSequence.html) against the regular expression. All of the state involved in performing a match resides in the matcher, so many matchers can share the same pattern.

A typical invocation sequence is thus Pattern p = Pattern.compile("a\*b"); Matcher m = p.matcher("aaaaab");

boolean b = m.matches(); A matches method is defined by this class as a convenience for when a regular expression is used just once. This method compiles an expression, that can be as simple as a normal string and matches an input sequence, e.g. a line from a binary or text file against it in a single invocation.

I use ArrayList’s of Strings to keep the (12) pattern strings and (7) Binary and Text Filenames.

In this project, files have to be in the default Netbeans working directory, otherwise on the directory that the code is running on. In another project, that I include here, I download the files from their URL that’s included in the file “urls.txt” to current default Netbeans project.

In yet another I read the pattern strings from the URL that keeps the keywords.txt Text File.

In the Output results I include Patterns searched(for every string pattern), Files Searched (for every file searched), Line where the matching occurs (if any), LineNumber, Position in line where the string occurs,

an image of the line, and Number of Occurrences per file involved.

Unit Testing should test the pattern word to be at the beginning/end of the line, surrounded by spaces, commas, not commas, punctuation, etc.), to look for the word inside another string.

Another very fast alternative approach (with Java), is to do the following: Convert the search string to byte-array, same encoding as the file. Open a memory mapped byte-buffer from File-Channel on the file. Scan the ByteBuffer, looking for matches to the search byte-array, count newlines as you go. At the end close the ByteBuffer.

If the file is larger than your memory, you will have to re-position the byte-buffer occasionally. Industry and literature recommend using memory-mapped size of about 4MB plus the size of the search-string. That way you can search the 4MB window, and then start the next window at the next 4mb boundary.

**SearchStringInTextFile,**

is another relevant project, where I examine alternative methodologies for searching strings in Text Files this time.

I use ArrayList<String> for the string patterns and I assume the files are on the default current working directory of Netbeans.

**DownloadFileFromURL,**

yet another project in which I use apache.commons.io.FilenameUtils to get the filename with the URL,

open URL connections, use DataInputStream, FileOutputStream to absorb the stream and write it to current default working root directory of the Netbeans project. If you don’t run it from Netbeans it downloads them in the current working directory where the jar file resides and runs.

**ReadTextLinesFromURL,**

In this project I navigate to a URL, I adopt BufferedReader, InputStreamReader to open the stream and read the Lines from the keywords.txt file. I then save them to an ArrayList<String> ready to be used by other classes waiting to receive them for searching purposes.