CS1003 – Programming with Data

Practical 4

Deadline: 23 April 2021, 9pm
Credits: 30% of coursework mark
MMS is the definitive source for deadline and credit details

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

This practical involves processing text files using Apache Spark and using basic text processing techniques to perform the specified task. You will need to decompose the problem into a number of methods as appropriate and classes if necessary. You will also need to test your solution carefully and write a report.

Task description

The task is to write a Java program to perform a fuzzy string search across a number of files. The search term will be given in the command line, together with the path to the directory that contains the text files and a *similarity threshold*. The program will print any matching results to standard output.

Command line arguments

1. Path to directory that contains a number of text files
2. The search term (typically multiple words in double quotes, like: "to be or not to be")
3. Similarity threshold (a floating-point number between 0 and 1).

For example, the following call should search in all text files stored at directory /path/to/datadir for the string "setting sail to the rising wind" with a similarity threshold of 0.75.

```
java CS1003P4 /path/to/datadir "setting sail to the rising wind" 0.75
```

The expected output using the 10 ebooks we provide is the following 3 results (in any order).

```
him setting sail to the rising
setting sail to the rising wind
sail to the rising wind the
```

Notice each result has 6 words (the same number of words as the search term) and their Jaccard similarity to the query is at least 0.75 (not displayed in the output).

See the following for some hints.

- A matching result will have the same number of words as the search term. Thus, you should check against all 4 word subsequences if the search string has 4 words in it.
- Use Jaccard similarity through 2-character bigrams to calculate similarity. This is very similar to what we did for Practical 1 (except no top-tail this time). You may reuse parts of your old code.
- The Jaccard similarity between the search term and the matching result should be greater or equal to the given similarity threshold.
- You should use JavaSparkContext.textFile or JavaSparkContext.wholeTextFiles to read the contents of the text files.
- This practical uses stacscheck. Have a look at how the tests are written and the expected outputs.

- Print one matching result per line to the standard output stream. The order of lines does not matter (stacscheck will check in an order independent manner).
- The data files we provide (in the Tests/data directory) are the most frequently downloaded ebooks in the last 30 days from Project Gutenberg at the time of writing.Code organisation
- Create a directory called CS1003-P4 and a file called CS1003P4.java inside of this directory.
- You may create more classes for organising your code. These should be stored in the same directory.
- The code will be compiled by running `javac *.java` in the CS1003-P4 directory. The Spark jar files will be added to the classpath. The jar files are stored in `/cs/studres/CS1003/0-General/spark`. You are not permitted to use any other external libraries.

Cleaning textual data

The textual data you read from the files needs to be cleaned before running a search on it. There are many ways of cleaning/normalising textual data, we chose a very straightforward definition in this practical.

- Remove all non-alphanumeric characters by replacing them with a space character.
  ```
  String text = …;
  text = text.replaceAll("[^a-zA-Z0-9]", " ");
  ```
- Convert the comment text to lowercase.
  ```
  text = text.toLowerCase()
  ```
- Split text on whitespace (space, tab and new line characters) to get an array of words.
  ```
  text = text.split("[ \t\n\r]")
  ```

Testing

This assignment makes use of the School's automated checker stacscheck. You should therefore ensure that your program can be tested using the auto-checker. This will permit you to see how well your program performs on the tests we have made public and will hopefully give you an insight into any issues prior to submission. The automated checking system is simple to run **on a lab computer** from the command line in your CS1003-P4 directory:

```
stacscheck /cs/studres/CS1003/Practicals/P4/Tests
```

You can run subsets of the test suite independently as well. Try the following commands:

```
stacscheck /cs/studres/CS1003/Practicals/P4/Tests/queries/sail
stacscheck /cs/studres/CS1003/Practicals/P4/Tests/queries/tree
```

Make sure to type the command exactly - occasionally copying and pasting from the PDF specification will not work correctly. If you are struggling to get it working, ask a demonstrator.

Use a lab computer for running stacscheck for consistency between the development environments.

The automated checking system will only check the basic operation of your program. It is up to you to provide evidence that you have thoroughly tested your program.

Submission

Your report must be structured as follows (a report template is provided on StudRes): + JavaDoc

- Overview: Give a short overview of the practical: what were you asked to do, and what did you achieve? Clearly list which parts you have completed, and to what extent.
- Design: Describe the design of your program. Justify the decisions you made. In particular, describe the classes you chose, the methods they contain, a brief explanation of why you designed your solution in the way that you did, and any interesting features of your Java implementation.
- Testing: Describe how you designed different tests and how you tested your program. Your report should include the output of your test runs to demonstrate that your program satisfies the specification. Please note that simply reporting the result of stacscheck is not enough; you should do further testing and explain in the report how you convinced yourself that your program works correctly. Do not submit screenshots, include text instead. Ideally your testing report should include a table of the following form:

What is being tested	**Name of the test method**	**Pre-conditions**	**Expected outcome**	**Actual outcome**	**Evidence**
…	…	…	…	…	…

- Evaluation: Evaluate the success of your program against what you were asked to do.
- Conclusion: Conclude by summarising what you achieved, what you found difficult, and what you would like to do given more time.

Don't forget to add a header including your matriculation number, the name of your tutor and the date.

Upload

Package up your CS1003-P4 folder and a PDF copy of your report into a zip file as in the previous practicals, and submit it using MMS, in the slot for Practical P4. After doing this, it is important to verify that you have uploaded your submission correctly by downloading it from MMS. You can then run stacscheck directly on your zip file to make sure that your code still passes stacscheck. For example, if your file is called 200000000-P4.zip, save it to your Downloads directory and run:

```
cd ~/Downloads
stacscheck --archive 200000000-P4.zip /cs/studres/CS1003/Practicals/P4/Tests
```

We covered using `scp` for copying files from your local computer to the lab computers and back.

Marking Rubric

1-6	Very little evidence of work, software which does not compile or run, or crashes before doing any useful work. You should seek help from your tutor immediately.
7-10	An acceptable attempt to complete the main task with serious problems such as not compiling or crashing often during execution.
11-13	A competent attempt to complete the main task. Serious weaknesses such as using wrong data types, poor code design, weak testing, or a weak report riddled with mistakes.
14-16	A good attempt to complete the main task together with good code design, testing and a report.
17-18	Evidence of an excellent submission with no serious defects, good testing, accompanied by an excellent report.
19-20	An exceptional submission. A correct implementation of the main task, extensive testing, accompanied by an excellent report. In addition, it goes beyond the basic specification in a way that demonstrates use of concepts covered in class and other concepts discovered through self-learning.

See also the standard mark descriptors in the School Student Handbook:
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

Lateness penalty

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good academic practice

The University policy on Good Academic Practice applies:
https://www.st-andrews.ac.uk/students/rules/academicpractice