Overview

This practical focused around streaming. We had to create a fuzzy search over files in a directory given. For the fuzzy string search itself, we were allowed to use the same jaccard-index code we created for P1. The search had to be made using Apache Spark, which can automate parallel processing, when setup correctly.

I was pleased with my resulting application, and completed all parts of the specification, to the fullest extent.

Finally, through this practical, I improved in my ability in designing applications with parallelisation in mind – something I will discuss further in the design section.

Design

The primary design decision for this practical, I felt, was in structuring the streaming process. I tried to look at all of the separate functions and steps, and look for where those tasks could be done independently of other tasks. Therefore, as each file could be independently searched, this could be done in a JavaRDD stream, where the contents is analysed.

I initially thought I may be able to extract each list of words of the same length of the query, and then stream over them to check for their Jaccard Index, but I realised that this would require looping over all possible "chunks" in a file, and collating them, before any actually get checked, and so I merged these processes together so the chunks get checked as they are created. This also saves on RAM requirements as you only need to temporarily save each chunk for that check, which gets overridden with the next chunk once finished. Simply, this results in matches being found as you go, and so the program could be ended earlier, if an early match is the one you wanted to find.

For retrieving the bigrams, I removed the addition of the $ and ^ characters from my practical 1 code, because with a fuzzy string search, it's looking for the set of words as a whole, and it would bias searches for chunks which begin and end with the same characters as the query which isn't what is desired.

Testing

What is being tested	Name of the test method	Pre-Conditions	Expected Outcome	Actual Outcome	Evidence
STACSCHECK	System Testing		All Tests Passed	All Tests Passed	$ stacscheck /cs/studres/CS1003/Practicals/P4/Tests/ Testing CS1003 CS1003 P4 - Looking for submission in a directory called 'CS1003-P4': Already in it! * BUILD TEST - build : pass * TEST - queries/advice/test : pass

					* TEST - queries/not-found/test : pass * TEST - queries/sail/test : pass * TEST - queries/tree/test : pass * TEST - queries/what-50/test : pass * TEST - queries/what-62/test : pass * TEST - queries/what-75/test : pass 8 out of 8 tests passed
User Input	Unit Testing	The folder "/does/not/exist" does not exist as a directory	Error message printed stating the directory doesn't exist	Error message printed stating the directory doesn't exist	$ java CS1003P4 /does/not/exist "test" 0.5 Directory doesn't exist: "/does/not/exist"
Data Directory User Input	Unit Testing	The folder "./empty-dir/ " exists, but contains nothing	No Output	No Output	$ java CS1003P4 ./empty-dir/ "test" 0.5 $
User Input – Threshold above 1	Unit Testing	The user inputs a threshold value greater than 1, but all other inputs are valid	Error message displayed stating the threshold must be between 0 and 1	Error message displayed stating the threshold must be between 0 and 1	$ java CS1003P4 ./data/ "test" -1 2>/dev/null Invalid Similarity Threshold: '-1' Must be a number from 0 to 1
User Input – Threshold below 0	Unit Testing	The user inputs a threshold value below 0, but all other inputs are valid	Error message displayed stating the threshold must be between 0 and 1	Error message displayed stating the threshold must be between 0 and 1	$ java CS1003P4 ./data/ "test" 2 Invalid Similarity Threshold: '-1' Must be a number from 0 to 1
User Input – Query is empty	Unit Testing	The user inputs an empty query string but all other inputs are valid	No Output	No Output	$ java CS1003P4 ./data/ "" 0.5 $
User Input – No Arguments Supplied	Unit Testing	The user doesn't give an input arguments	Error message displayed giving the number of arguments given, required and a brief usage guide	Error message displayed giving the number of arguments given, required and a brief usage guide	$ java CS1003P4 2>/dev/null ERROR: 0 / 3 parameters given Usage: java CS1003P4 search-directory "query" similarity-threshold
User Input – Less Than 3 Arguments Given	Unit Testing	The user gives more than 0, but less than 3 arguments	Error message displayed giving the number of arguments given, required and a brief usage guide	Error message displayed giving the number of arguments given, required and a brief usage guide	$ java CS1003P4 "./data/" ERROR: 1 / 3 parameters given Usage: java CS1003P4 search-directory "query" similarity-threshold
User Input – More Than 3 Arguments Given	Unit Testing	The user gives more than 3 arguments	Error message displayed giving the number of arguments given, required and a brief usage guide	Error message displayed giving the number of arguments given, required and a brief usage guide	$ java CS1003P4 "./data/" "test" 0.5 "test" ERROR: 4 / 3 parameters given Usage: java CS1003P4 search-directory "query" similarity-threshold
File Reading – Only Empty Data Files	Unit Testing	The "./data/" directory given only contains empty files	No Output	No Output	$ java CS1003P4 ./data/ "" 0.5 $

File Reading – The Data Directory Contains a Directory	Unit Testing	The "./data/" directory contains another directory	The directory is ignored, but no error message is given	The directory is ignored, but no error message is given	$ mkdir ./data/test-dir
					$ java CS1003P4 ./data/ "test sample with no matches" 1
					$

Evaluation

In implementing my design, I originally did use the $ and ^ characters, but took bigrams for each individual word in the chunks and query. With this, I realised that this would require much more processing, and also prioritised shorter words such as “it”, “is”, “an”, and “the”. Therefore, I removed this, and just took bigrams from the string as a whole, including the single spaces between words.

Overall, I am particularly pleased with the design and implementation of my program, with an efficient search technique. I applied parallelisation in an efficient way, and I feel I used streaming tools in an appropriate manner, with each function being kept as independent as possible.

Conclusions

To conclude, I feel like my application could be adapted for use in a file system, such as “FZF”, if I also printed the line-numbers, and file names of the matches.

It may have also made the application more useful if the matching chunks were merged where they overlap with other chunks, because this would make understanding where matches are easier.

Given more time, it would have been interesting to implement an option for recursive file searching, meaning folders in the data file could be searched. This would have brought up interesting issues with massive file directory trees, but may be solved with the implementation of clean early termination of the program, once the user is happy with the current matches.