Overview

For this practical we were required to use the Hadoop implementation of MapReduce to manipulate XML data files. The program should first obtain the absolute paths of the input and output directories from the user. Next the XML data files must be read in from specified input directory, and the output must be written to a new file created in the specified output directory. The output file should contain all character level or word level n-grams for text fragments contained within files in the given input directory.
The program should only take in the text fragments written in a given language which is decided depending on the user's input and thus the output file should could the list of n-grams and their frequency in the input language.

Design

In order to complete this practical, I have decided to use the latest version of the Hadoop API over the old API. There are multiple reasons for doing so but put simply the new API cleaner and and has enabled me to write the MapReduce job in a more convenient, easier and sophisticated fashion.
When I first attempted this practical I decided to use flags in the mapper class in order to find the lines that where between certain tags in the input XML document. However, this method was not only inefficient but could also cause problems when not working with a local-standalone, pseudo-distributed of the Hadoop installation.
On fully-distributed Hadoop installation the data is sent to multiple nodes, and processed in parallel. Consequently, in simple terms, this could mean that at one node a flag is set whereas in another node the flag is not. This would result in incorrect data being retrieved from the input file, or worse still data being missed completely.
Because of this I have also decided to define my own input format for the MapReduce jobs as when you get a different types of files for processing you have to create your own custom input format for processing using MapReduce jobs.
This has enabled me to extract the correct data from the xml files and then map the data accordingly. The custom input format works in a similar fashion to the input file formats found in the Hadoop API e.g. TextInputFormat. So although there is no format for XML documents which is the format that the input data is in I have used Mahout's XMLInputFormat code. The code is Licensed to the Apache Software Foundation and openly available on GitHub (https://github.com/apache/mahout/blob/66f164057e322d2e63ea02c35c9e30c3969e80b1/integration/src/main/java/org/apache/mahout/text/wikipedia/XmlInputFormat.java).
The Mahout's XMLInputFormat code takes care of figuring out the record boundaries with in your XML input files using the specified start and end tags.

For this practical I have decided to create a GUI, this is simply because it enforces validation and verification and generally makes the program more user friendly.

When the program is first run an instance of the GUI class initiated on a new thread and the graphical user interface is displayed. The GUI is relatively basic and only contains a few components. The first components are the 'jButtonGetInputFile' and the 'jButtonGetOutputFile' buttons. These buttons are used to allow the user to select the directory where the input data is located and the directory where the output data should be stored. To ensure valid directories are input I have used a 'JFileChooser' component, which will force the user to select a folder, and the directory of the selected folder is stored. If a directory is not selected a warning message is displayed and the program will not continue. For ease of use I have set the default directory of the JFileChooser component to the directory the program is located in. Next I have used two combo boxes, one which contains all the langue's that can be selected and one which contains numbers from 1 – 20 for the word/char gram length. These therefore means the user cannot input an invalid language or word/char gram length. Next, there are two radio buttons which are linked (only one can be pressed at any given time) which allow the user to select if they want to find word grams or char grams. The

next component is the “jCheckBoxSortAlphabetically” check box, if selected the output of the program will be sorted alphabetically, however if it is not selected the program will use the “myInverseMapper” switch the key from word to frequency and sort the output by frequency instead. Finally, the ‘jToggleButtonSearch’ button is used to send the data input via the GUI to the main MapReduce program.
Once the MapReduce program has created the output file, the contents of the output file are displayed in the ‘jTextAreaOut’ component and message is displayed notifying the user where the output file is located. The reason I diced to read in the contents of the output file into the ‘jTextAreaOut’ is because the data is then ordered by count, if I where the add the data at an earlier stage, the lines would no longer be ordered.

Once the user has input the required information via the GUI, the main class is used to create the map reduce job. First of all, a new configuration object used to store the input information, this information can then be retried elsewhere. Next the the first job is created which is used to map frequencies to words, the input format for this job is set to the ‘XMLInputFormat’ class.

The ‘XMLInputFormat’ class is used only to create a ‘XMLRecordReader’ object. In the ‘XMLReaderClass’ the XML input files are taken in as an input. Next the start tag if defined, using the language input by the user and the end tag is also defined. These Strings are then encoded into a sequence of bytes using "utf-8", storing the result into a new byte array. Because start tag appears multiple times in each file, there are multiple start location stored. Next the program finds the position of the first byte in the file to process, using the encoded start tag. The number of bytes in the file to process is calculated in order for the program to know where to stop.
The XmlInputFormat will seek to byte address/offset of the block / split boundaries and then scan forwards until it finds either the configured XML start tag or reaches the byte address of the block/split boundary. If it finds the start tag, it will then consume data until it finds the end tag (or end of file). If it finds the end tag a record will be passed to the mapper, otherwise the mapper will not receive any input.

The ‘MyMapper’ class is the first mapper that I use. Here a new ‘InputStream’ object is created which reads the next byte of the data from the the input stream and returns an integer in the range of 0 to 255. If no byte is available because the end of the stream has been reached. Next a ‘DocumentBuilderFactory’ object is created which is a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents. A ‘DocumentBuilder’ object is then used to parses the contents of the given input source as an XML document and returns a new DOM Document object. Finally, ‘Document’ and ‘NodeList’ objects are created and nodes are returned in the order in which they are specified in the XML document.
A for loop is used to loop through the nodes returned and new element containing the current node is made. This element contains the data retired from the XML file.
Because the element could still contain unwanted data e.g. punctuation, I have used the java regex “[^\p{L}]” which keeps all Unicode characters from various alphabets, not just the Latin alphabet. All blank spaces are then replaced by a single space and each separate word contained in the element object is added to a list.
Finally based on whether the user selected word level n-grams or char level n-grams, the map function maps each each n-gram with a frequency of one. I have used code from my Practical 2 in order to find word and char grams from a list of words.
Once the mapper has then finished, the reducer class is used to, where the key is the word the values are all the counts associated with that word (one copy of '1' for each occurrence).
If the user has selected to to sort the data alphabetically, the program finishes here and the output is stored in the selected output directory.

If the user however has not selected sort alphabetically then the output of the reducer is saved in an "intermediate" folder in the project directory. Following this the class "MyInverseMapper" is used to order the data held in the intermediate folder to order the n-grams by occurrence frequency.

Finally it should be noted that I have used the apache.commons.io.FileUtils API to allow the program to automatically delete the "intermediate" folder created and also the selected "output". This is necessary in order for the program to create them again.

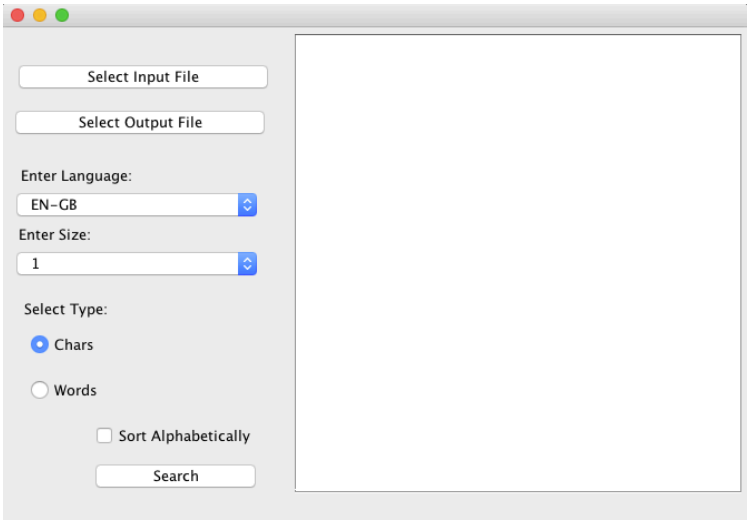
Extensions

I've implemented the following extensions:

- Graphical User Interface
- Custom defined XML format
- Order the n-grams by occurrence frequency
- Automatically delete files and Using the latest Hadoop API. **

Examples

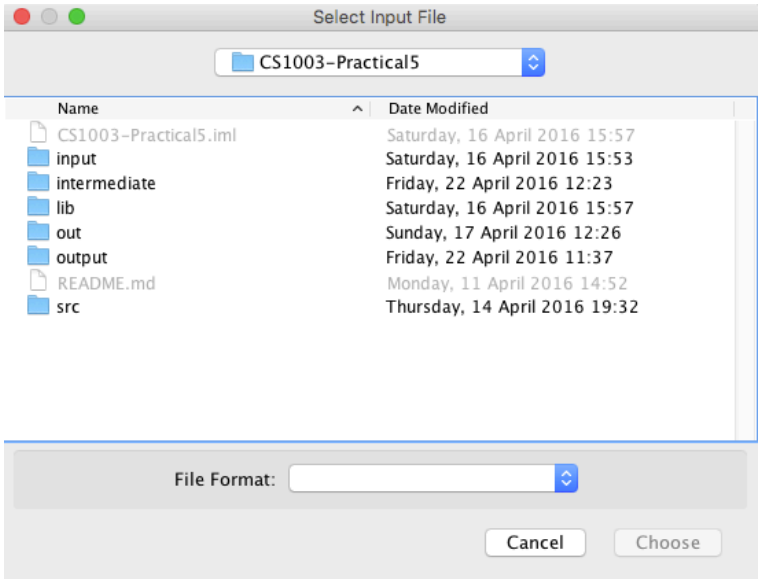
Start screen:



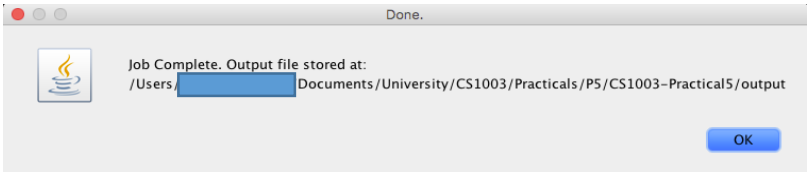
All the components are initialized and have default values where appropriate.

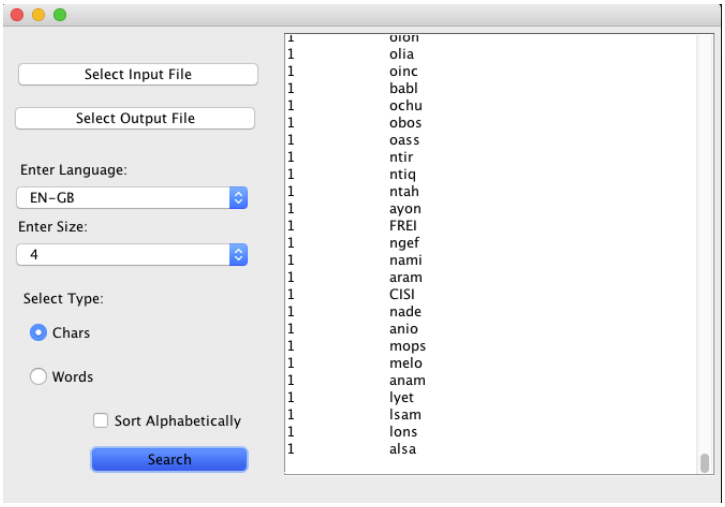
Select File Screen:

The director selector, files are greyed out as they cannot be selected only folders can



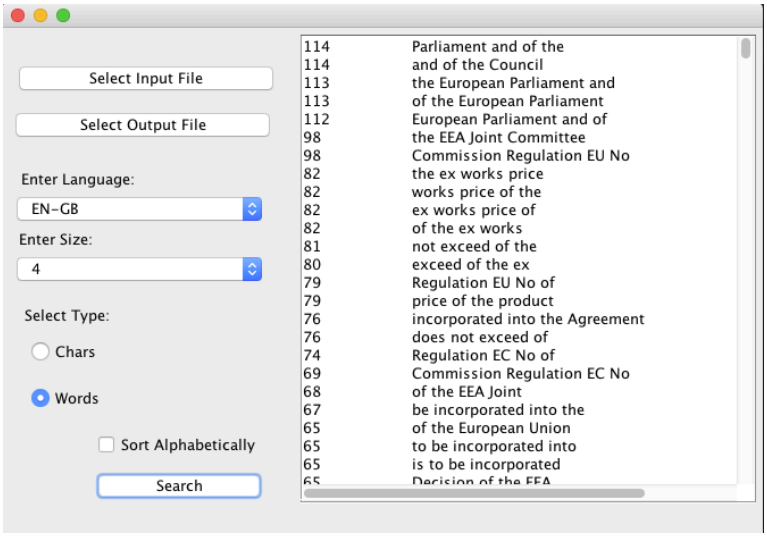
Output (sorted by frequency of occurrences)





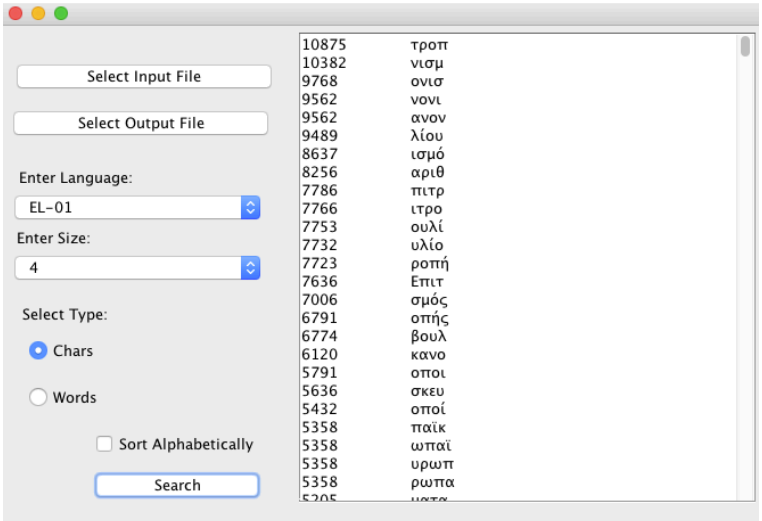
The pop up message contains the output directory and the text area is populated Character level n-grams. They are correctly displayed and ordered by frequency of occurrences

Word level n-grams output(sorted by frequency of occurrences):



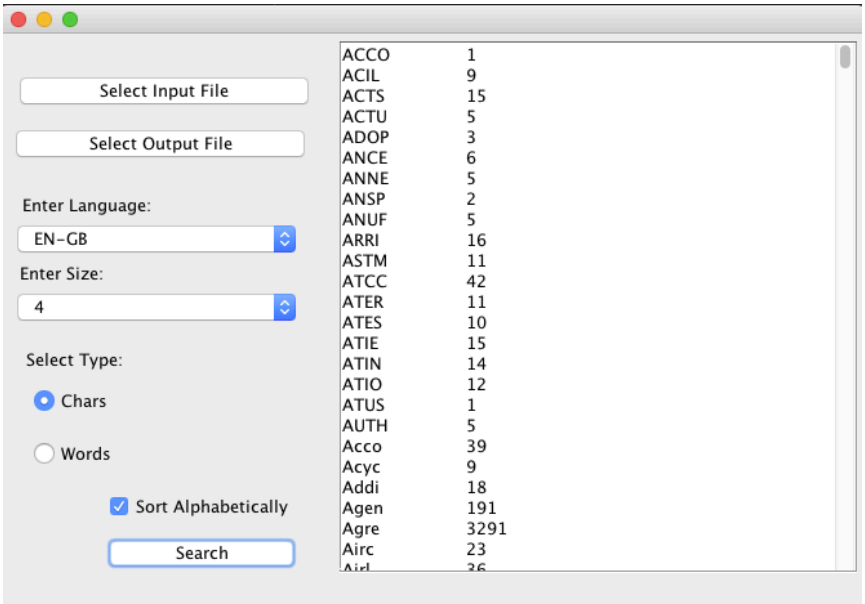
A pop up message contains the output directory and the text area is populated word level n-grams. They are correctly displayed and ordered by frequency of occurrences

Non-Latin alphabet output:



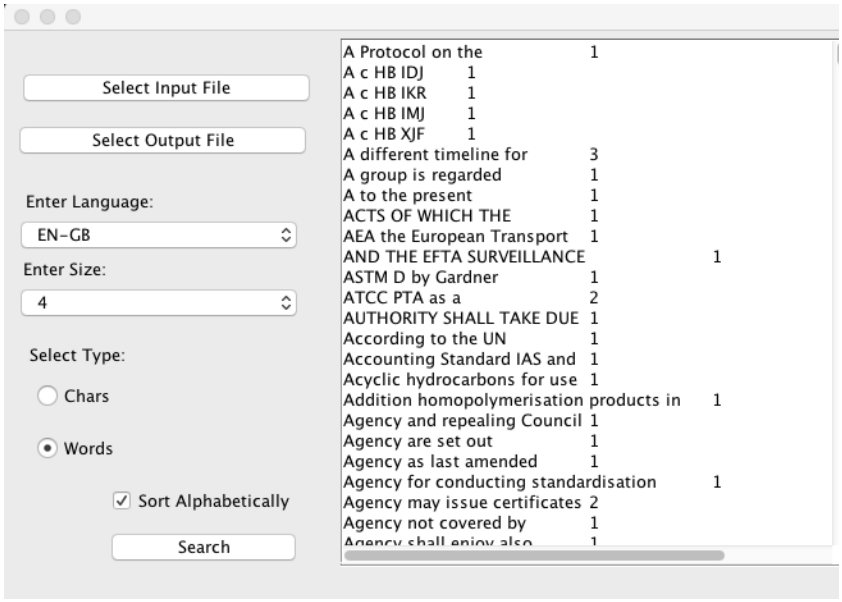
A pop up message contains the output directory and the text area is populated by Latin and not also non-Latin alphabet characters but no other non-letter characters are displayed

Char level n-grams output(sorted alphabetically):



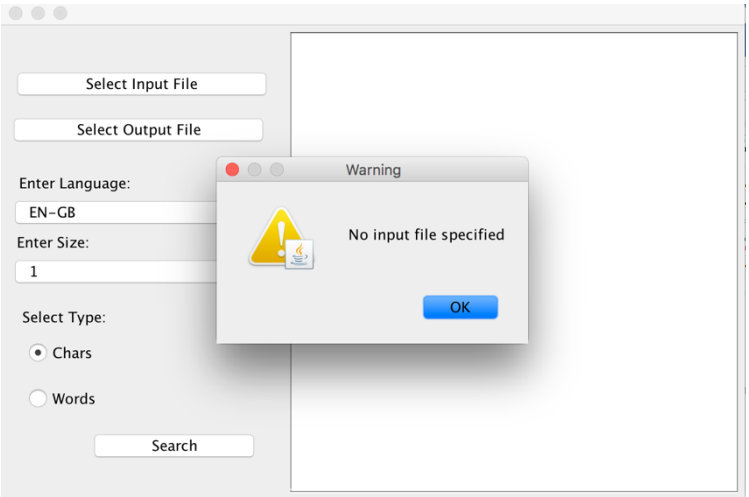
A pop up message contains the output directory and the text area is populated character level n-grams. They are correctly displayed and ordered alphabetically

Word level n-grams output (sorted alphabetically):



A pop up message contains the output directory and the text area is populated word level n-grams. They are correctly displayed and ordered alphabetically

No file error:



If no input file or output file is selected the following warning message is displayed

Testing

Test Description	Expected Output	Actual Output	Comment
Char level grams, Length 1	The program will correctly add every separate character from all the input files to the output file, the frequencies will be calculated correctly and the text area on the GUI will displayed the ordered results	The program correctly found and added every separate character from all the input files to the output file, the frequencies were calculated correctly and the text area on the GUI did display the ordered results	The program worked as expected
Char level grams, Length 5	The program will correctly find every 5 adjacent characters from all the input files to the output file, the frequencies will be calculated correctly and the text area on the GUI will displayed the ordered results	The program correctly found and added every 5 adjacent characters from all the input files to the output file, the frequencies were calculated correctly and the text area on the GUI did display the ordered results	The program worked as expected
Word level grams, Length 1	The program will correctly add every separate word from all the input files to the output file, the frequencies will be calculated correctly and the text area on the GUI will displayed the ordered results	The program correctly found and added every separate word from all the input files to the output file, the frequencies were calculated correctly and the text area on the GUI did display the ordered results	The program worked as expected
Word level grams, Length 5	The program will correctly find every 5 adjacent words from all the input files to the output file, the frequencies will be calculated correctly and the text area on the GUI will displayed the ordered results	The program correctly found and added every 5 adjacent words from all the input files to the output file, the frequencies were calculated correctly and the text area on the GUI did display the ordered results	The program worked as expected

Sort alphabetically	The data in the output file and the data displayed in the text area should be sorted alphabetically and the key should be the word	The data in the output file and the data displayed in the text area was sorted alphabetically and the key was be the word	The program worked as expected
Sort by frequency of occurrences	The data in the output file and the data displayed in the text area should be sorted by frequency of occurrences and the key should be the frequency of occurrences	The data in the output file and the data displayed in the text area was sorted by frequency of occurrences and the key was the frequency of occurrences	The program worked as expected
No input directory selected	The GUI should display a message box notifying the user no input folder has been selected	The GUI displayed a message box notifying the user no input folder was selected	The program worked as expected
No output directory selected	The GUI should display a message box notifying the user no output folder has been selected	The GUI displayed a message box notifying the user no output folder was selected	The program worked as expected
Select empty input directory	The program should continue as usual but the output file will not contain any data	The program run as it usually would and finished without any errors but the output file was empty	The program worked as expected
Input file not in XML format	The program should continue as usual but the output file will not contain any data	The program run as it usually would and finished without any errors but the output file was empty	The program worked as expected
30 input files (large no. of input files)	The program should continue as usual, the processing time should also not be effected	The program continued as usual, and the processing time was not effected	The program worked as expected

Evaluation

The program I designed meets all the initial requirements successfully as well as incorporating various different extensions. After extensively testing the core functionality of the program, I'm are pleased to report that the program works as expected and is capable of handling erroneous conditions with ease. Moreover, the program gracefully displays the output results and is user friendly whilst still being quick and efficient.

Conclusion

In conclusion I'm satisfied with end program, it is able to carry out the required tasks with ease and is able to handle erroneous data without crashing. The way the program has been designed means that breaking the program is challenging. The added extensions massively increase the overall functionality.

If we were to use a fully-distributed Hadoop installation, then using Hadoop MapReduce would be a very quick an efficient way of processing large amount of data in parallel. Although the traditional method of file processing we used in earlier practical's are somewhat easier to understand and simple, Hadoop's MapReduce would be the obvious choice for processing large amounts of data. Moreover, Hadoop's MapReduce API has a wide range of functionality and allows user to manipulate data in various different ways. There are some draw backs to MapReduce though, for example having limited input formats, however this is not a huge issue, as there is, as I found, a large number of openly available resources online, which provide good quality solutions to such problems.