CS1003 Practical 2 150007691 Tutor: Shyam Reyal 26th Feb 2016

Overview:

In this practical I was asked to write a Java program that would read a text file and count the number of occurrences of all the possible groups of three consecutive characters, called trigrams. Punctuation and numerals were to be ignored, and letters were to be converted to lower case. The end of a word was to be represented with an underscore. The program then had to sort the trigrams by number of occurrences and list the top ten most frequent trigrams in a CSV file.
For the extensions I created a UI, the ability to get grams of different numbers, the ability to get word grams and a simple word predictor.

Design:

- Try, catch and finally blocks where used throughout the program were necessary to prevent the crashing of the program if the file was missing or did not have the right permissions.
- A finally block was added to allow for the writer and reader to close even if there was an exception in the try block.
- I used a while loop to read in all the data from the the file. This allows for different sized files to be read with this program. The data was saved in array list form as I do not know the amount of data that is being input.
- Static variables where used to hold the data that was read in to allow for use of the data throughout many methods.
- A trigram class was created to all the storage of object trigrams that had a trigramString and number of occurrences. The total number of trigrams was also stored as a static field.
- The program makes the user enter the file path for a folder of files that they want to be included in the menu of the program. If the user enters the wrong path the program will tell them and gracefully quit.
- The program allows the user to enter numbers to choose options on the menu, the program validates their in put well.
- The fileFinder method creates an array of file paths for all the text files in the input folder to allow the user to choose between them.
- The readFile method allows the user to choose a file to read in and then the number of characters they want the grams to have. The program then reads through the file picking out all valid grams and putting in them into an array list.
- The program checks if the same gram has already been found before, if it has then it will just add one to the trigramCount attribute. This was the most efficient method of storing the grams, it did not require the initial storage of invalid grams. It is all validated at once.
- The program then sorts the grams by descending order of number of occurrences using the Collections.sort() method. This method had to be overridden as I did not want to compare trigram objects but I wanted to compare the trigram objects through their attributes.
- The readWordsFromFile() and createWordsFromFile() methods do largely the same thing as the readFile() method except that it reads in full words. Instead of choosing the number of characters in the gram the user chooses the number of words in a row for each gram. The words are output in the right format with a "_" for the end of words.
- The readWordFromFilePred() method reads in one word grams from the file, it does not let the user choose the number of words. This is because we use the one word grams to predict the mostly word when half of the word is typed. Fairly simple but effective.
- The writeToCSVFile() method outputs the grams array list to a csv file in correct csv format. A for loop is used for efficiency.
- The textPrediction() method allows a user to enter part of a word. The program will then run through the sorted array list for grams that are at least two characters longer than the input (this allows for at least one character and "_" for the end of the word). If it finds one then it

will check if the input string is the same as the start of the gram. If it isn't then it will move onto the next gram in the array list. But if it does match then the program will output that gram as the most likely word to be entered. This is very efficient because as soon as the the first match is found the program will not search any more of the list as it does not need to.

- The program will also tell the user if a match was not found.
- For the prediction extension I take in data from all three text files in the input files folder. I did not have time to program to allow user to choose what files to put in. The program will only work for this extension if there are exactly three text files in the input folder.

Testing:
Testing was fairly easy for this practical. I just had to open the csv file and see if the totals when in descending order. I also checked the text files with control f to see if there was about the same number of words. I also tried breaking the program by not putting any files in and putting in the wrong file path but the program handles this with ease. I tested the predictive text creating my own custom one gram csv file and trying some of the more popular words.

Evaluation:
My program met all the requirements and passed all my tests. I would evaluate this program as a success, the error handling means that I will be hard to break the program. The program is also very efficient as it uses loops where possible.

Conclusion:
I achieved a fully working program that met all the requirements and handles errors very well, I also managed to finish all the extensions suggested. I found it difficult sometimes to set up the for loop counters to select the data from my array lists as I wanted it to be able to do it with different files. If I had more time I would've made all parts of the program (reading in the files and sorting the data for character word and predictive text) run through less methods. There is code duplication for the extensions as the loops had to be slightly different but if I had a few more days I think could've made it work with duplicating any code.