COSC4315/COSC6345: Functional programming
Finding the k least frequent keywords in a file

1 Introduction

You will write a program to find the most important words in a file: keywords. This is the main mechanism to find documents on the web with a search engine like Google. There will be an input integer k to find the k least frequent words.

The input files will be small, relative to RAM. Since this homework requires functional programming, you must use recursive functions and lambda expressions to solve the problem. You should worry less about memory efficiency and time complexity, but try to reach good O().

The programming language will be: JavaScript (aka JS). The interpreter will be 'node'.

2 Input

The input file name will be different in every call: it will be the main input parameter. Therefore, your program will have to parse the input file name.

The input is one text file, with at most 1000 different words or numbers, separated by spaces, punctuation symbols and any other common symbols. Repeated spaces, tabs and new lines should be considered as one space. Words will be either English words, either from a dictionary, acronyms (e.g. DNA, IP, RAID) or names (city, person, company). You can assume words will be at most 20 characters long, combining lower and uppercase.

3 Program and input specification

The main program should be called "keyword" (exactly like this, in lower case, please avoid other names).

Call syntax, assuming your main program is 'keyword.js', is as follows:

```
node keyword.js "input=xyz.txt;k=3;output=keywords.txt"
```

A word is regular expression of letters, starting with a letter, perhaps combined with digits (dictionary words or variable names or product names), where letters can be lower or upper case. Names and common words are all letters. Similary, a number is a regular expression of digits, perhaps having a decimal part. Notice that words can be followed by punctuation signs anywhere. Words can be capitalized (e.g. names).

4 Output

The output should be sent to the output file *exactly* in the format below with each word and its frequency, one per line. Avoid changing the output format or adding any other messages since this will prevent automated testing.

Output example with $k = 2$ and least frequent keywords:

```
Gramercy 3
Bellafontaine 2
```

5 Requirements

- Programming language: JavaScript, meeting the requirements below.
- **Functional programming** is the main requirement, as detailed below. Basically, this means all functions should be recursive, there should be no variable mutation and all computations will happen via lists.
- **Recursion** is required. You can develop forward or backward recursion starting from either side of the list. You can get 10% extra credit for forward recursion for a maximum grade of 100 (cannot be accumulated towards other HW or exam).
- Functional expressions: Your program should internally use lambda expressions when possible. Your program should use map()/reduce() functional constructs to compute frequencies, not simple linear recursion.
- Lists (dynamically sized) are required to store words: input, temporary and output. You cannot use existing libraries with arrays, dictionary or hash tables data structures. Therefore, you cannot use their associated functions either. Using such libraries defeats a functional programming approach. Make sure lists are not mutated after function calls.
- Recursive functions are required to process lists. Specifically, it is unacceptable to use traditional loops (while/for) to process the list(s). While() or for() loops are acceptable, only to read the input file into a list or produce the final output file. Nevertheless, recursive functions to read and write files are encouraged.
- There will be a list of stopwords (articles, adverbs, prepositions) that should be eliminated (provided by the instructor). These words provide no value in web searches. The remaining words are important. Those are the keywords, which are the main output.
- Search and sort algorithms: All lists should be processed with recursive sort and recursive search functions, without mutation. It is acceptable to have an $O(n^2)$ sort algorithm, but $O(n \log(n))$ is encouraged; in your source code comments specify which sort/search algorithm you are using. Search can also be either sequential or binary, with binary being preferred. Keep in mind a recursive sort algorithm will use significant RAM. Notice you cannot use built-in sort functions since they produce variable mutation. Therefore, you have to develop your own sort and search functions.
- In this HW edition numbers can be ignored.

- Rank ties should be considered (i.e. keywords having the same frequency). All of them should be included and must be sorted alphabetically. This will be a rare case, but you should still consider it.

- Correctness is the most important requirement: TEST your program with many expressions. Your program should not crash or produce exceptions with any input or wrong parameters.

- Functional, structured programming is required. The list needs to be passed as function argument and/or function return value, but cannot be a global variable. The program must be prepared to handle zeroes or simple errors like spaces or missing numbers.

- Create a README file explaining how to run your program and explaining any issues or incomplete features.

- Your program will be tested for originality, comparing it to other students source code, source code students from past semesters and top source code sites from the Internet (e.g. github, stackoverflow). You must disclose any source code that was not written by you. Keep in mind several students may get source code from the same site, especially if it comes as a top hit in Google.

- Your program should display a small "help" if not input parameters are provided and must write error messages to the screen. Your program should not crash, halt unexpectedly or produce unhandled exceptions. Consider empty input, blank lines and so on.

- Unix system (UH Linux server). You can develop and test your code on any computer or operating system (e.g. Windows, Mac), BUT your program will be tested only on our course server. Therefore, test it carefully on the UH server before the deadline.

- Source code:

  the quality of your source code will be graded including using theory learned in class, clarity of identifiers, indentation, comments, elegance of recursive solution. Moreover, source code originality will be checked with plagiarism tools. Notice that source code written by one person, later modified by another person, is likely to be detected.

- Test cases:

  Your program will be tested with 10 test cases, going from easy to difficult. You can assume 70% of test cases will be clean, valid input files. The remaining 30% will test correctness and efficiency with harder input. We will avoid abnormal, superhard test cases (e.g. words with 100 letters, file with 1M words, etc). Nevertheless, try to handle such rare cases. The TA will make *preliminary* test cases available 1 week before the deadline so that you can self-test your program; please report any issues with the test cases as soon as possible in case the TA has made any mistake or some test case is too difficult.

- Grading:

  A program not submitted by the deadline is zero points Any exception or late submission must be discussed and approved by the instructor at least 2 days before the deadline. Notice this HW is impossible to do in a couple of days. A non-working program, failing to produce output, is worth 10 points (i.e. not producing output for the easiest test case). A program that does some computations correctly, but fails several test cases (especially the easy ones) is worth 50 points. Only programs that work correctly with most input files that produce

correct results will get a PASS score of 60 or higher. In general, correctness is more important than speed.

- Resubmission: For programs with a score 80 or less, there will be one opportunity to fix a program and resubmit within 3 days. minor fixes are 10%, no matter how small (all of them together 10% based on the opinion of the TA and the instructor). major fixes are 30% off (something that requires hours of work, based on the instructor opinion). Programs with a score 85 or higher can be resubmitted, but the additional credit will be considered only at the end of the course.