CSCI 4963/6963 — Large-Scale Programming and Testing
Homework 2 (document version 2.3)
Text Analysis and Inverted Indexes in C

Overview

- This homework is due by 11:59:59 PM on Friday, November 10, 2017.
- This homework will count as 10% of your final course grade.
- This homework is to be completed **individually**. Do not share your code with anyone else.
- You **must** use C for this homework assignment, and your code **must** successfully compile via `gcc` with absolutely no warning messages when the `-Wall` (i.e., warn all) compiler option is used. We will also use `-Werror`, which will treat all warnings as critical errors.
- Your code **must** successfully compile and run on Submitty, which uses Ubuntu v16.04.3 LTS. Note that the `gcc` compiler is version 5.4.0 (`Ubuntu 5.4.0-6ubuntu1~16.04.4`).

Homework Specifications

In this second homework, you will build on your first homework assignment by using C to analyze text, calculate a variety of statistics, and construct an inverted index. In brief, your program must apply your regular expression logic from the first homework assignment to a specified list of input files; for each file, you will use regular expressions to identify all words, bigrams, and trigrams. Further, all identified words, bigrams, and trigrams will be indexed (and therefore searchable).

Using Regular Expressions to Identify Words

For this assignment, assume you are processing HTML documents, though your program should also work for general text-only documents. Similar to the `wordfreq.php` example, perform the following preprocessing before you identify words:

1. Remove or ignore any scripts, including all text within `<script>` tags. **(v1.2)** Note that `<script>` tag content could span multiple lines.
2. Remove or ignore all HTML tags. **(v1.2)** Note that this include closing tags (e.g., `</title>`, `</h2>`, etc.) and abbreviated tags (e.g., `<br/>`, `<img ... />`, etc.).
3. Remove or ignore any special characters; **(v1.2)** the list of special characters for this assignment are: `&nbsp;`; `&quot;`; `&amp;`; `&lt;`; and `&gt;`.

Words must be identified as consisting of alpha characters (i.e., as identified by the `isalpha()` function of the `ctype.h` library). Further, for this assignment, convert everything to lowercase.

Also, to capture contractions (e.g., "don't"), allow single quotes to be part of a word, though words must always begin and end with a letter (**v1.2**). Further, only one single quote can be in a word; therefore, a subsequent single quote would act as a word delimiter.

(v1.2) You can continue to take a line-based approach for all of the above except for the `<script>` tag content.

Identifying Bigrams and Trigrams

In addition to identifying individual words, you must also identify bigrams (e.g., United States) and trigrams (e.g., Rensselaer Polytechnic Institute). **(v1.2)** Note that bigrams and trigrams may span multiple lines. Unlike individual words, the most frequently occurring bigrams and trigrams are deemed to be of interest. Therefore, you will need to capture and index the top 20 bigrams and the top 10 trigrams from the given set of input files.

Extending the Regular Expression Language

Two key updates to the first homework assignment are required.

First, you must support bracket expressions as follows. Any set of verbatim characters or regex characters (i.e., `.`, `\d`, `\w`, and `\s`) can be combined within square brackets to indicate that one character must be matched in this group. Note that a repetition symbol (i.e., `*`, `+`, or `?`) may be applied to the entire group. Also note that groups within groups is not allowed.

Some example regular expressions to support (and test for) are:

```
 [aeiou]           match a single vowel character
 part [123]        match "part 1" or "part 2" or "part 3"
 part [12][ab]     match "part 1a" or "part 1b" or "part 2a" or "part 2b"
 [\w\d]            match a single character that is either an alpha
                    character or a digit
 [\w\d]+           match a word consisting of alpha characters and/or digits
```

Second, you must provide a uniform interface (i.e., function) in C for your regular expression code. The function prototype (and requirements) are summarized below.

```
/*  regex_match() applies the given regex to each line of the file
 *   specified by filename; all matching lines are stored in a
 *    dynamically allocated array called matches; note that it is
 *     up to the calling process to free the allocated memory here
 *
 *  regex_match() returns the number of lines that matched (zero
 *   or more) or -1 if an error occurred
 */
int regex_match( const char * filename, const char * regex, char *** matches );
```

Calculating Text Statistics

Using whatever data structures you like, keep track of all words that you read in from each file (i.e., document). You will need to keep track of the number of individual word occurrences, the number of bigrams, the number of trigrams, the total number of documents, and the total number of unique words, bigrams, and trigrams.

Note that before you identify bigrams and trigrams, you first must identify the top 50 individual words **(v1.7)** across all documents. These words must then be skipped as you identify the "interesting" bigrams and trigrams (or else you will end up with many useless results).

After you have processed all of the specified documents, output the most frequently occurring words, bigrams, and trigrams as shown in the example below. Sort your output by most frequently occurring to least frequently occurring.

(v1.7) For any ties, sort by the order in which you discovered the words, bigrams, or trigrams. More specifically, sort by document order (based on the given input file) and the order in which words first appear in the given document(s). In other words, the assumption here is that you are processing the document from beginning to end, detecting words as you go.

As an example, if you run your program with only the `book-1984.txt` example, your output would be as follows:

```
Total number of documents: 1
Total number of words: 104420
Total number of unique words: 8910
Total number of interesting bigrams: 25923
Total number of unique interesting bigrams: 20829
Total number of interesting trigrams: 11356
Total number of unique interesting trigrams: 10942

Top 50 words:
6522 the
3493 of
2574 a
2445 and
2348 to
2316 was
...

Top 20 interesting bigrams:
72 big brother
40 thought police
36 old man
30 inner party
...

Top 10 interesting trigrams:
13 two minutes hate
11 three super states
8 junior anti sex
...
```

Building an Inverted Index

The last stage of this assignment is to build a rudimentary inverted index. For each word, bigram, and trigram, you will need to keep track of which document(s) they appear in, as well as the number of occurrences.

For example, if the word “beautiful” appears 17 times in document A and 12 times in document B, your data structure would keep track of the following:

```
beautiful: [ doc A, 17 ], [ doc B, 12 ]
```

You can use whatever data structure(s) you like here; just be sure to match the output requirements described below.

Once you have processed all of the specified documents, output your inverted index to the file specified by the second command-line argument. Each line of the output file must be formatted as follows:

```
<word-or-bigram-or-trigram>: [ <docID>, <count> ], [ <docID>, <count> ], etc.
```

Here, words, bigrams, and trigrams are all combined together into one large list. Further, you must alphabetize your output. And note that `docID` values are positive integers assigned to each document in the order given in the input file.

Command-Line Arguments

The first command-line argument specifies the name of a text file containing a list of input files to be read and processed. Each line contains a single filename or path; in other words, assume that either an absolute or relative path is used here.

The second command-line argument specifies the name of the output file to write your inverted index to.

Implementation Details

As with the first homework assignment, for your implementation, consider using `fopen()`, `fscanf()`, `fgets()`, `fclose()`, `isalpha()`, `isdigit()`, `isspace()`, and other such string and character functions. For the `regex_match()` function, use `calloc()`. Be sure to check out the details of each function by reviewing the corresponding `man` pages.

If improper command-line arguments are given, report an error message to `stderr` and abort further program execution. In general, if an error is encountered, display a meaningful error message on `stderr` by using either `perror()` or `fprintf()`, then aborting further program execution.

Error messages must be one line only and use the following format:

```
ERROR: <error-text-here>
```

Assumptions

For this assignment, you can make the following assumptions:

1. Assume that each specified filename is no more than `128` characters.
2. Assume that the maximum line length for any input file is **(v1.6)** `1024` characters.

Submission Instructions

To submit your assignment (and also perform final testing of your code), please use Submitty, the homework submission server. The specific URL is on the course website.

Note that this assignment will be available on Submitty a few days before the due date. Please do not ask on Piazza when Submitty will be available, as you should perform adequate testing on your own.

(v1.5) To ensure that your `regex_match()` function can be automatically tested on Submitty, please enclose your `main()` function with an `#ifndef`, as shown below:

```
#ifndef USE_SUBMITTY_MAIN
int main( ... )
{
  ...
}
#endif
```

To make sure that your program does execute properly everywhere, including Submitty, read the following tip.

Output to standard output (`stdout`) is buffered. To ensure buffered output is properly flushed to a file for grading on Submitty, use `fflush()` after every set of `printf()` statements, as follows:

```
printf( ... );     /* print something out to stdout */
fflush( stdout );  /* make sure that the output is sent to a */
                   /*  redirected output file, if specified  */
```