MATLAB Programming NYCU Fall 2024 LAB#6 2024/10/24

This lab includes a little of string processing, file I/O, and MATLAB structures. You should write a function that takes a path to a text file, find the individual words in the text, determine the unique words and count their times of occurrence, and organize the information of the words in a MATLAB structure array that is sorted according to a given criterion.

For this lab, you can use any string processing function provided by MATLAB.

- The function:

  ```
  A = my_word_count(fn, sort_mode)
  ```

  Here **`fn`** is the path to the file. The output **`A`** is an array of structures with three fields: **`word`** is the word itself (make it a string scalar or character vector), **`len`** is the length of the word, and **`count`** is its times of occurrence. Example:

  `A(1).word` is `'she'`, **`A(1).len`** is 3, and **`A(1).count`** is `4`.

  `A(2).word` is `'sells'`, **`A(2).len`** is 5, and **`A(2).count`** is `4`.

  The second input **`sort_mode`** is a string scalar or character vector that indicates how the words should be sorted. It can be **`'word+'`** or **`'word-'`**, meaning sorting in dictionary order; **`'len+'`** or **`'len-'`**, meaning sorting by length; **`'count+'`** or **`'count-'`**, meaning sorting by count. Here the part of **`'+'`** or **`'-'`** in **`sort_mode`** indicates that the sorting is in the ascending or descending direction, respectively.

  Finally, if you call the function with no output argument, let the function print out the words and their lengths and counts in the given sorted order. You should still use the structure **`A`** internally.

- Sorting:

  You can use **`[A.field]`** to put all the values of the same field in the structure array into a vector, and sort this vector. This applies to **`[A.len]`** and **`[A.count]`**. For sorting the words, the method depends on whether your words are strings or character vectors. If they are strings, you can directly use **`[A.word]`** for sorting. If they are character vectors, you need to use **`{A.word}`** to put all the words into a cell array of character vectors, and they apply **`sort`** to this cell array.

  Since you cannot sort **`A`** directly, you need to use sort to get the "sorted index" first, and then use the index to rearrange the elements of **`A`**:

  ```
  [~, k] = sort([A.field]);
  A = A(k);
  ```

- Text Processing:

  First use a text editor to save the following text (which is an English tongue-twister) to a text file, and use its path as the input to your function:

  ```
  She sells seashells by the seashore.
  The shells she sells are surely seashells.
  So if she sells shells on the seashore,
  I am sure she sells seashore shells.
  ```

  There are several different ways to retrieve the words from the file. The most convenient approach here is to use **`textscan`**:

  ```
  s = textscan(fid, '%s' ,'delimiter', '., ');
  W = s{1};
  ```

  Here **`fid`** is the file ID returned by **`fopen`**. Remember to close the file after reading its content.

  The function **`textscan`** returns a cell array where each element corresponds to one item in the format specification. That is why we have to take just its first element, which is itself a cell array that contains all the individual words. The use of the **`delimiter`** attribute is a convenient way to directly skip the white spaces and punctuations when reading the file.

Convert all the words to lower case using the function **`lower`**. Tip: Many MATLAB string processing functions can be directly applied to all the elements in a cell array of character vectors or a string array in a single call. This reduces the need to use loops and leads to more compact and efficient programs.

You can apply **`unique`** to a cell array of character vectors or a string array. Use this technique to get the unique words. Then loop over your list of unique words to determine their counts of occurrence.