Questions and Solutions

May 27, 2020

1 Questions and Solutions

1.0.1 Introduction:

The code of these three questions are all written in Python. For each question, I created some test case to test the function.

```
In [50]: import sys
         print('Python Version:')
         print (sys.version)
```

```
Python Version:
3.6.8 |Anaconda, Inc.| (default, Dec 29 2018, 19:04:46)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
```

1.1 Problem 1

You are given a string, and the task is to implement a function that finds the longest unrepeated string. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1. For "ababzab", it is "abz" and the length is 3. The string contains only lower-cased letters, i.e., 'a'-'z'.

```
In [12]: def findLongestUnrepeatedSubstring(s):
             l = len(s)  # length of the input string
             max_l = 0  # record the maximum length of substring without repeating characters
             longest_unrepeated_string = ''
             start_index = 0  # the start point of substring
             chars_index = dict()  # record the index of the chars

             # iterate over the input string
             for i in range(l):
                 # check if the char already in the substring
                 if s[i] not in chars_index:
                     chars_index[s[i]] = [i]  # store its index into dict
                     if i - start_index + 1 > max_l:  # update the maximum length
                         max_l = i - start_index + 1
                         longest_unrepeated_string = s[start_index:i + 1]
```

```
                else:
                    # if the last same num is in front of the start point
                    if chars_index[s[i]][-1] < start_index:
                        if i - start_index + 1 > max_l:  # update the maximum length
                            max_l = i - start_index + 1
                            longest_unrepeated_string = s[start_index:i + 1]
                    else:
                        # if this char already in chars_index dict, find its last index in st
                        last_same_char_index = chars_index[s[i]][-1]
                        # update the star point of the substring
                        start_index = last_same_char_index + 1
                    # record the index of the char
                    chars_index[s[i]].append(i)

            return longest_unrepeated_string
```

1.1.1 Test

```
In [13]: strings = ['abcabcbb', 'bbbbb', 'ababzab', '', 'abcdefg', 'bbtablud']
         for string in strings:
             print(string + ":  " + findLongestUnrepeatedSubstring(string))
```

```
abcabcbb:  abc
bbbbb:  b
ababzab:  abz
:
abcdefg:  abcdefg
bbtablud:  tablud
```

1.2 Problem 2

(Please write this one in C# if you know C#) Write a program that counts word occurrences in a directory of UTF-8-encoded text files and outputs a file (UTF-8) containing a list sorted (a) by frequency and (b) alphabetically within the same frequency. Words should be normalized to lower case (i.e., "Hat", "hAt" and "HAT" should all be normalized to "hat"). You can assume words are space-separated tokens. The directory may contain sub directories and we need to search through all sub directories. Also, the directory may contain different type of files, but we should only process files with extension ".txt".

If your input directory contained a single file that looks like this:

```
The goal of the game is to win the game.
```

… then the output of your program should be a file that looks like this:

```
the 3
game 2
goal 1
```

```
is 1
of 1
to 1
win 1
```

You can use any standard library classes and methods that are part of the C# language.

```
In [20]: import os
         import string

         def wordsFrequencyCounter(directory):
             # get all of the directories and files
             paths = os.listdir(directory)
             # initial an empty dict
             words_freq = dict()
             # iterate all of the paths
             for path in paths:
                 # check if the path is a directory
                 if os.path.isdir(directory + '/' + path):
                     words_freq = mergeDictionaries(words_freq, wordsFrequencyCounter(director
                 # check if the path is a file whose type is txt
                 if os.path.isfile(directory + '/' + path) and path.endswith('.txt'):
                     words_freq = mergeDictionaries(words_freq, readFile(directory + '/' + pat
             return words_freq


         def readFile(path):
             # read the text from the file
             text = open(path, "r").read()
             # normalize words to lower case
             text = text.lower()
             # remove punctuations
             text = text.translate(str.maketrans('', '', string.punctuation))
             # split the string and count the frequency of each word
             words = text.split(" ")
             words_freq = dict()
             for word in words:
                 if word not in words_freq:
                     words_freq[word] = 1
                 else:
                     words_freq[word] += 1
             return words_freq


         def writeFile(path, words_freq):
             file = open(path, 'a')
             for key in sorted(words_freq, key=words_freq.get, reverse=True):
                 file.write( key + ' ' + str(words_freq[key]) + '\n')
```

```
                print(key + ' ' + str(words_freq[key]) + '\n')
            print("File is saved")


        def mergeDictionaries(dict1, dict2):
            for key in dict2:
                if key not in dict1:
                    dict1[key] = dict2[key]
                else:
                    dict1[key] += dict2[key]
            return dict1
```

1.2.1 Test

The structure of the directories

```
.
 Coding Questions
     files for question 2
         sub directory 1
             sub directory 3
                 sub directory 4
                     test_text_3
             test_text_2
         sub directory 2
         test_text_1
     Code.py
     Questions and Solutions.ipynb
```

```
In [47]: current_directory = os.path.dirname(os.path.realpath('__file__'))

         words_freq = wordsFrequencyCounter(current_directory + '/' + 'files for question 2')

         d_words_freq_file = current_directory + '/' + 'words frequency.txt'

         writeFile(d_words_freq_file, words_freq)
```

```
the 9

game 6

goal 3

of 3

is 3

to 3
```

```
win 3

most 1

important 1

File is saved
```

1.3 Problem 3

Complete a function to find the sub array of an integer array with largest sum. For example, for the array [1, -2, 3, 10, -4, 7, 2, -5], you should find 3 + 10 + (-4) + 7 + 2 = 18. It is best to write the code in C#. You can reuse the attached .cs file.

```
In [31]: def maxSumOfSubarray(nums):
             current_max_sum = list()
             current_max_sum.append(nums[0])
             for i in range(1, len(nums)):
                 current_max_sum.append(max(current_max_sum[i - 1] + nums[i], nums[i]))
             return max(current_max_sum)
```

1.3.1 Test

```
In [46]: arrays = list()
         arrays.append([1, -2, 3, 10, -4, 7, 2, -5])
         arrays.append([8,-19,5,-4,20])
         arrays.append([-2,1,-3,4,-1,2,1,-5,4])
         arrays.append([3,-2,-3,-3,1,3,0])
         arrays.append([-2,-1])
         arrays.append([2,-2,-2,0,-2,2,2])

         for array in arrays:
             print(str(array) + ': ' + str(maxSumOfSubarray(array)))
```

```
[1, -2, 3, 10, -4, 7, 2, -5]: 18
[8, -19, 5, -4, 20]: 21
[-2, 1, -3, 4, -1, 2, 1, -5, 4]: 6
[3, -2, -3, -3, 1, 3, 0]: 4
[-2, -1]: -1
[2, -2, -2, 0, -2, 2, 2]: 4
```