CSCI 150: Exam 3 Practice

November 1, 2021

The in-class practice is the Extra Tracing Optional Homework. For the take-home part, write code which accomplishes each of the following:

1. Write a function `string_min` which takes in a string `s` of lower case characters and returns the *index* of the character closest to the front of the alphabet. You can assume that `s` will always be non-empty. If the minimal character appears more than once, the first index (i.e. lowest number) should be returned.

   For example:

   - `string_min('catalog')` should return `1` (since 'a' is the lowest character and its first occurrence is in index 1.)
   - `string_min('computer')` should return `0`
   - `string_min('string')` should return `5`
   - `string_min('movie')` should return `4`

2. The five English language vowels are *a*, *e*, *i*, *o*, and *u*. (We will ignore *y*, sorry). Write a function `vowel_counter` which takes in a string parameter `s` and returns a dictionary, whose keys are the vowels which appear in the string and values are the number of times they appear. Though the input might mix upper and lower case, you should lower-ize, so that, say 'e' and 'E' should be treated as two occurrences of 'e'. The string might have punctuation, spaces, and other special characters.

   For example:

   - `vowel_count('catalog')` should return `{'a': 2, 'o': 1}` (notice that 'e', 'i', and 'u' are not keys, since they do not appear in the word)
   - `string_min('A man. A plan. A canal. Panama')` should return `{'a': 10}`
   - `string_min('computer')` should return `{'o': 1, 'u': 1, 'e': 1}`
   - `string_min('The quick brown fox jumps over the lazy dog!')` should return `{'e': 3, 'u': 2, 'i': 1, 'o': 4, 'a': 1}`
   - `string_min('Try!')` should return `{}`

- `string_min('')` should return `{}`

3. You will be given a dictionary as a parameter. The dictionary will have as keys single lower-case letters and values non-negative integers. Write a function `str_build` which will return a string of the keys of that dictionary which each key is repeated the number of times its integer value counts.

   For example:

   - `str_build({'g': 3, 'a': 2, 'x': 5})` will return `'aagggxxxxx'`
   - `str_build({'g': 3, 'x': 5, 'a': 2})` will return `'gggaaxxxxx'`
   - `str_build({})` will return `''`
   - `str_build({'z' : 10, 'b' : 2, 't': 7})` will return `'zzzzzzzzzzbbttttttt'`

4. This problem has two parts:

   (a) Write a function `div_2` which takes a positive integer parameter `n` (you do not need to worry that it will ever be 0 or negative) and returns the number of *integer* divisions by 2 that need to occur to produce an answer ≤ 1.

   For example:

   - `div_2(10)` should return `3`, since the division sequence is $10 \rightarrow 5 \rightarrow 2 \rightarrow 1$
   - `div_2(1)` should return `0`, since no division are needed
   - `div_2(3)` should return `0`, since the division sequence is $3 \rightarrow 1$

   (b) Now, write a function `all_div2` which takes in a single positive integer parameter `n` (again, always certain to be positive) and:

   - will determine the output from `div_2` for each integer between 1 and `n`, including `n` itself
   - will organize this information into a dictionary, keyed on the number of steps returned by `div_2` and with value a list of the integers that took that number of steps.

   For example:

   - `all_div2(5)` will return `{0: [1], 1: [2, 3], 2: [4, 5]}`
   - `all_div2(9)` will return `{0: [1], 1: [2, 3], 2: [4, 5, 6, 7], 3: [8, 9]}`