COMPUTER PROJECT 9

Assignment Overview

This program focuses on the use of dictionaries, sets, files and text manipulation.

This assignment is worth 65 points (6.5% of course grade), and must be submitted before **11:59 PM on Monday, April 17, 2023**. After the due date, 1 pt will be deducted for every 1 hr. late. Note that you need to click on the submit button to be considered as submitted.

Background

We take the word completion for granted. Our phones, text editors, and word processing programs all provide suggestions for how to complete words as we type based on the letters typed so far. These hints help speed up user input and eliminate common typographical mistakes (but can also be frustrating when the tool insists on completing a word that you don't want completed).

Overview

You will implement two functions that such tools might use to provide command completion. The first function, `fill_completions`, will construct a dictionary designed to permit easy calculation of possible word completions. A problem for any such function is what *vocabulary*, or set of words, to allow completion on. Because the vocabulary you want may depend on the domain a tool is used in, you will provide `fill_completions` with a representative sample of documents from which it will build the completions dictionary (we provide the file `ap_docs.txt`). The second function, `find_completions`, will return the set of possible completions for a prefix of any word in the vocabulary (or the empty set if there are none). In addition to these two functions, you need a function to open a file, and will implement a simple driver program to use for testing your functions.

Program specifications

`Open_file () → file pointer:`

- This function prompts the user to enter a vocabulary file name. The program will try to open the data file. Appropriate error message should be shown if the data file cannot be opened. This function will loop until it receives proper input and successfully opens the file. Use of `try-except` is required. It returns a file pointer. Use this function from previous projects. Check the `strings.txt` file for the appropriate strings to use in the prompt and error message to match the tests. Use the `'UTF-8'` encoding when opening the file. For example:

```
fp = open("melody.txt", encoding='UTF-8')
```

a. Parameters: `no parameters`
b. Returns: a file pointer
c. Display: prompts and error messages

`read_file (fp) → set of words:`

a. This function uses the provided file pointer and reads the vocabulary data file. It returns a set of words in the file:
    1. Words are stripped of punctuation. Use `string.punctuation`
    2. Words containing non-alphabetic characters are ignored (e.g., `didn't` or `m'abandonne` or `Est-ce`), as are words of length 1 (since there is no reason to complete the latter).

For example, if we use the `melody.txt`, the function will return a set that is equivalent to the following set since order in sets does not matter:

```
{'tom', 'such', 'sun', 'tow', 'toot', 'suit', 'son', 'sofa', 'too', 'soffit',
'town', 'to'}
```

b. Parameters: `file pointer`
c. Returns: set of words
d. Display: nothing

`fill_completions(words)` → `dictionary`:

- This function takes a set of words and returns a dictionary whose keys are `tuples` and the values are `sets`. It returns the completion dictionary as described below.
    - The keys are tuples of the form `(n, l)` for a non-negative integer `n` and a lower-case letter `l`:
      `word_D[(i,ch)] = set of words with ch at index i`
    - The value associated with key `(n, l)` is the **set** of words in the file that contain the letter `l` at position `n`. For simplicity, all words are converted to lower case. For example, if the file contains the word `"Python"` then the sets returned by `word_D[0,"p"], word_D[1,"y"], word_D[2,"t"], word_D[3,"h"], word_D[4,"o"],` and `word_D[5,"n"]` all contain the word `"python"` (as well as other words).

  For example,

  `Words =` `{'tom', 'such', 'sun', 'tow', 'toot', 'suit', 'son', 'sofa', 'too', 'soffit', 'town', 'to'}`

  The function will return the following dictionary:

```
{(0, 't'): {'too', 'to', 'town', 'tow', 'toot', 'tom'},
(1, 'o'): {'too', 'sofa', 'to', 'soffit', 'town', 'tow', 'son', 'toot',
'tom'},
(2, 'o'): {'too', 'toot'},
(0, 's'): {'sofa', 'soffit', 'son', 'such', 'suit', 'sun'},
(2, 'f'): {'soffit', 'sofa'},
(3, 'a'): {'sofa'},
(3, 'f'): {'soffit'},
(4, 'i'): {'soffit'},
(5, 't'): {'soffit'},
(2, 'w'): {'town', 'tow'},
(3, 'n'): {'town'},
(2, 'n'): {'son', 'sun'},
(1, 'u'): {'such', 'suit', 'sun'},
(2, 'c'): {'such'},
(3, 'h'): {'such'},
(2, 'i'): {'suit'},
(3, 't'): {'toot', 'suit'},
(2, 'm'): {'tom'}
}
```

- Remember that the order in a set does not matter. Two sets that are out of order are equivalent. So, if your sets have a different order than the ones shown in this document or in the test cases, your test did not fail because of the order.

a. Parameters: `set of words`
b. Returns: `dictionary`
c. Display: nothing

`find_completions(prefix, word_D)` → `set of strings`:

- This function takes a `prefix` of a word (possibly empty) and a completions dictionary of the form described above (`word_D`). It returns the **set** of words in the completions dictionary that complete the prefix, if any. If the prefix cannot be completed to any vocabulary words, the function returns the empty **set**.
  For example,
  `word_D =`
```
{(0, 't'): {'too', 'to', 'town', 'tow', 'toot', 'tom'},
(1, 'o'): {'too', 'sofa', 'to', 'soffit', 'town', 'tow', 'son', 'toot',
'tom'},
(2, 'o'): {'too', 'toot'},
(0, 's'): {'sofa', 'soffit', 'son', 'such', 'suit', 'sun'},
(2, 'f'): {'soffit', 'sofa'},
(3, 'a'): {'sofa'},
(3, 'f'): {'soffit'},
(4, 'i'): {'soffit'},
(5, 't'): {'soffit'},
(2, 'w'): {'town', 'tow'},
(3, 'n'): {'town'},
(2, 'n'): {'son', 'sun'},
(1, 'u'): {'such', 'suit', 'sun'},
(2, 'c'): {'such'},
(3, 'h'): {'such'},
(2, 'i'): {'suit'},
(3, 't'): {'toot', 'suit'},
(2, 'm'): {'tom'}
}
```

  `prefix = 'sof'`

  the function will return the following set:
  `{'sofa', 'soffit'}`
  Which is the intersection between the following 3 sets:
  `word_D[(0, 's')]`, `word_D[(1, 'o')]`, and `word_D[(2, 'f')]`

a. Parameters: `str, dictionary`
b. Returns: `set of strings`
c. Display: nothing

`main()`

- Main part of your program:
  - Calls `open_file()` to get a file pointer.
  - Calls `read_file()` to get the set of words from the file
  - Calls `fill_completions` to fill out a completions dictionary using this file. Sort the completions in alphabetical order and print them separated by a comma and space. Do not leave a trailing comma. Hint: build a string and slice off the trailing comma & space or use `join()`.
  - Repeatedly prompts the user for a prefix to complete or for an `'#'` to quit.
  - Prints the set of words that can complete each prefix or states that the prefix has no completions.

Assignment Deliverable:

The deliverable for this assignment is the following file:

`proj09.py` -- your source code solution

Be sure to use the specified file name and to submit it for grading via the **Coding Rooms system** before the project deadline.

Assignment Notes:

0. Items 1-9 of the Coding Standard will be enforced for this project.
1. You will find `enumerate` very useful, e.g.
   ```
   for i,ch in enumerate(someString):
   ```
2. The design of the completions dictionary makes retrieval of completions a simple matter using intersection of sets. Consider, for example, possible completions of `"pyt"`. You should be able to convince yourself that the set of possible completions is the **intersection** of the sets `word_D[0,"p"], word_D[1,"y"],` and `word_D[2,"t"]`.
3. Python provides a binary operation for finding intersection of sets, denoted &. However, you need to form the intersection of an arbitrary number of sets (depending on the length of the prefix). How will you do this? In principle, this is no different than finding the sum of the numbers in a list `L` of arbitrary size using the binary + operator. To sum the list you initialize a working variable, say `result`, making it 0. Then, you add each subsequent element of `L` to `result` possibly using `"+="`, You can do essentially the same thing for the intersection problem. For the example above, you can initialize `result` to be `word_D[0,"p"]`. Then, using **set intersection** on `result` and `word_D[1,"y"]` to get the next value for `result`. Finally, intersect `result` and `word_D[2,"t"]` to get the intersection of the three sets.

Sample Output 1

```
Input a file name: xxyy

[Error]: no such file

Input a file name: test2.txt

Enter a prefix (# to quit): sof

The words that completes sof are: sofa, soffit

Enter a prefix (# to quit): to

The words that completes to are: to, tom, too, toot, tow, town

Enter a prefix (# to quit):

There are no completions.

Enter a prefix (# to quit): abc

There are no completions.

Enter a prefix (# to quit): #

Bye
```

Sample Output 2

```
Input a file name: ap_docs.txt

Enter a prefix (# to quit): prompt

The words that completes prompt are: prompt, prompted, prompting

Enter a prefix (# to quit): col

The words that completes col are: col, cold, coleman, collapse, collapsed,
collateral, colleages, colleague, colleagues, collected, collecting, college,
colleges, collett, collision, colman, colo, colombia, colombian, colombo,
colonial, colony, colorado, coloradoan, colorblind, colossus, colter, columbia,
column, columnist

Enter a prefix (# to quit): weig

The words that completes weig are: weigh, weighed, weighs, weight

Enter a prefix (# to quit): abc

The words that completes abc are: abc

Enter a prefix (# to quit): x

The words that completes x are: xinhua, xxx
```

```
Enter a prefix (# to quit): covid

There are no completions.

Enter a prefix (# to quit): #

Bye
```

Grading Rubric

```
Computer Project #09                                          Scoring Summary
General Requirements:
  ( 5 pts) Coding Standard 1-9
      (descriptive comments, function headers, mnemonic identifiers, format,
      etc...)

Implementation:
 ( 5 pts)  open_file function (no tests)

 ( 10 pts)  read_file function

 ( 12 pts)  fill_completions function

 ( 12 pts)  find_completions function

 ( 5 pts)  Test 1

 ( 5 pts)  Test 2

 ( 5 pts)  Test 3

 ( 4 pts)  Test 4

 ( 4 pts)  Test 5


Note: hard coding an answer earns zero points for the whole project
-10 points for not using main()
```