CSE 231 Spring 2013

COMPUTER PROJECT #8

Assignment Overview

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

This assignment is worth 50 points (5% of course grade), and must be submitted before 11:59 PM on Monday, March 18th .

Background

Nowadays we take word completion for granted. Our phones, text editors, and word processing programs all give us 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. The second function, `find_completions`, will return the set of possible completions for a start of any word in the vocabulary (or the empty set if there are none). In addition to these two functions, you will implement a simple `main` program to use for testing your functions.

Program specifications

- `fill_completions(c_dict, fd)` returns `None`. This function takes as input a (blank) completions dictionary and an opened file. It returns nothing, but it fills in 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`.
    - 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 vocabulary words are converted to lower case. For example, if the file contains the word `"Python"` then the sets returned by `c_dict[0,"p"]`, `c_dict[1,"y"]`, `c_dict[2,"t"]`, `c_dict[3,"h"]`, `c_dict[4,"o"]`, and `c_dict[5,"n"]` all contain the word `"Python"`.
    - Words are stripped of punctuation.
    - "Words" containing non-alphabetic characters are ignored, as are words of length 1 (since there is no reason to complete the latter).
- `find_completions(prefix, c_dict)` returns a set of strings (`str`). This function takes a (potential) prefix of a vocabulary word and a completions dictionary of the form described above. It returns the set of vocabulary words in the completions dictionary, if

any, that complete the prefix. It the prefix cannot be completed to any vocabulary words, the function returns the empty set.

- `main()`, the test driver:
  - Opens a file named "`ap_docs.txt`". This file contains a collection of old newswire articles. Each article in the collection is separated by a line that contains only the token "`<NEW DOCUMENT>`". We don't care about the distinction between documents for the purpose of building a completions dictionary. So you do not have to do anything special with these lines.
  - Calls `fill_completions` to fill out a completions dictionary using this file.
  - 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 that prefix has no completions.

Deliverables:

`proj08.py` -- your source code solution

1. Please be sure to use the specified file name, and to submit your file for grading via the "handin" program (http://www.cse.msu.edu/handin/webclient).

2. Save your file to your CSE disk (H drive on CSE computers) -- archiving files on the CSE disk is the only evidence that you have completed the project, if something has gone wrong. Get a TA to help you do this, if you do not know how.

Assignment Notes:

0. Be smart!!! Implement the functions and test them thoroughly on inputs for which you know what the answer should be. That means you will want to use a much smaller input file initially. Do the `main` function last, as this ties everything together.
1. This assignment makes extensive use of mutable arguments. Changes that a function makes to a mutable argument affect the value of the mutable argument in the caller. The function `fill_completions` takes advantage of this property of mutable arguments to fill in the completions dictionary. But this property can produce some surprising effects when the caller does not expect a function to change the values of its arguments, as in the case of `find_completions`. You will need to be careful not to inadvertently modify the completions dictionary that is passed in as the second parameter in the body of `find_completions`.
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`". Using discrete math (yeah CSE 260!), you should be able to convince yourself that the set of possible completions is the intersection of the sets `c_dict[0,"p"]`, `c_dict[1,"y"]`, and `c_dict[2,"t"]`.
3. Python provides a binary operation for finding intersection of sets, denoted `&`. But 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. For the summation problem, you initialize a working variable, say `result`, making it 0, if the list is empty or `l[0]`, otherwise. Then, you add each subsequent element of `l` to `result`. You can do essentially the same thing for the intersection problem. For the example above, you can initialize `result` to be `c_dict[0,"p"]`. Then, intersect `result` and `c_dict[1,"y"]` to get the next (a more accurate) value for `result`. Finally, intersect `result` and `c_dict[2,"t"]` to get the intersection of the three sets.

Questions for you to consider (not hand in):

A problem with your `find_completions` function is that, for many short word prefixes, it returns too many possible completions to be useful. An editing tool that invokes your function will need to select some subset of the possible completions to display to a user. To permit this, it would be useful if your `find_completions` function returned a ranked list of completions, in order by decreasing frequency of use. If you assume that the input file to your `fill_completions` function is representative for the domain of the tool you are building, this function could collect the information needed to determine how to rank the possible completions for each prefix. How would you redesign the completions dictionary to record this information? How would you modify your two functions? Finally, is it better for `find_completions` to return a ranked list of all possible completions for a prefix or just the five or six top-ranked words? Why?