FIT2004: Assessment questions for week 10

THIS PRAC IS ASSESSED! (5 Marks)

DEADLINE: Sunday, 06-May-2018 23:55:00
LATE PENALTY: 25% per day
MARKING: You will be interviewed during your lab or at another time as arranged by your demonstrator who will ask you a series of questions to assess your understanding of this exercise, and gauge how you implemented it. It is required that you implement this exercise strictly using **Python programming language**. Practical work is marked on the complexity of your program **and also on your understanding of the program**. A *perfect* program with zero understanding implies you will get **zero** marks! "Forgetting" is not an acceptable explanation for lack of understanding. Demonstrators are not obliged to mark programs that do not run or that crash.
You will lose all marks for a given task if your algorithm does not produce correct results for the task. Also, heavy penalties are applied if your complexity is worse than the required complexity and you may lose all marks if the complexity is similar to that of the naive algorithm. This is because our focus is on developing efficient algorithms and data structures. Also, do not assume that each task carries equal marks.
SUBMISSION REQUIREMENT: You will need to submit a zipped file containing your Python program (named autocomplete.py) as well as a PDF file briefly describing your solution and its space and time complexity. The PDF file must give an outline of your solution (e.g., a high level idea of how did you solve it) and the **worst-case** space and time complexity of your solution. Penalties will be applied if you fail to submit the PDF file. The zipped file is to be submitted on Moodle before the deadline.
Important: The assignments will be checked for plagiarism using an advanced plagiarism detector and the students will be interviewed by tutors to demonstrate the understanding of their code. Last year, many students were detected by the plagiarism detector and almost all got zero mark for the assignment and, as a result, many failed the unit. "Helping" others is NOT okay. Please do not share your solutions with others. If someone asks you for help, ask them to visit us during consultation hours for help.

1 Customized Auto-complete

Alice used to love the auto-complete feature in her phone until the day she sent her thesis advisor an email saying "Thank you for being on my adversary panel". She soon received a reply "I hope you meant the advisory panel ;)". Needless to say, she was embarrassed and wrote a long email (this time disabling the auto-complete feature) explaining that the mistake was caused due to the auto-complete feature. She decided to do something about it and now wants to modify the auto-complete feature in her phone.

She has come to you for help once again and asks, "Can you please help me implement a customized auto-complete feature for my phone?"

"Well, I would love to help but I am quite busy with studies and ...", you murmur. Alice says, "Oh come on! I will provide you all the data, and all you have to do is to implement an algorithm". You desperately try to explain your situation, "Alice! We have a mid-semester test in week 8. I need to start preparing for that".

Alice does not give up so easily, "Firstly, you should have started preparing for the test a couple of weeks ago. Secondly, I can wait till the end of week 9. Thirdly, implementing the algorithm will strengthen your understanding which will help you in the mid-semester test. Finally, I know you are such a good friend!!! Pleaseeeeeeeee :)". And before you could say something, she starts explaining the customized auto-complete feature.

The standard auto-complete function in her phone displays up to three words that have the user's entered word as a prefix. E.g., if you are typing "adv", the auto-complete feature shows "adversary", "advisory" and "adventure". She wants to modify the auto-complete feature such that it only shows one word (instead of three) but also shows the definition of the word so that she can make sure that she is using the right word. She also wants to know how many words are there in the dictionary that have this prefix. E.g., if there are 43 words that have "adv" as a prefix, the auto-complete feature must display 43.

She has downloaded a dictionary from the internet that contains English words and their definitions. She has also downloaded the text of all the emails she has ever sent. She has processed the text and has assigned each word in the dictionary a frequency which corresponds to the number of times she ever used the word in her emails. She wants the auto-complete feature to display the word in the dictionary that has the highest frequency among the words that have her entered word as a prefix. For example, assume that the frequencies of "adversary", "advisory", and "adventure" are 100, 200 and 150, respectively. If she types "adv", the modified auto-complete system should display "advisory" with its definition and the number 43 that corresponds to the number of words that have "adv" as a prefix . She implemented a sorting based algorithm to do the prefix matching but it is too slow. You need to help her in efficient implementation of this feature.

You are given a text file `Dictionary.txt` that contains English words, their frequencies and their definitions. You must write a Python program named `autocomplete.py` that reads from the input file and creates a Trie. Once the Trie has been constructed, the program will ask the user to enter a prefix. The program must then display the prefix matched word having the highest frequency along with its definition and the number of words in the dictionary that have this prefix.

1.1 Input

Input is the file `Dictionary.txt` that contains around 3,000 words, their frequencies and their definitions (see Moodle). However, below we illustrate the input and output assuming that the dictionary contains the details of only the following four words.

```
word: align
frequency: 358
definition: To adjust or form to a line; to range or form in line; to bring
into line; to aline.

word: adversary
frequency: 157
definition: One who is turned against another or others with a design to
oppose

word: advisory
frequency: 720
definition: Having power to advise; containing advice; as, an advisory
council; their opinion is merely advisory.

word: adventure
frequency: 696
definition: To try the chance; to take the risk.
```

Although `Dictionary.txt` provided on Moodle is sorted in alphabetical order of words to make it easier for you to manually verify the correctness of your implementation (and contains only the words starting with letter a), you cannot assume that the file on which your algorithm will be tested will also be sorted and will be similarly small. Your program may be tested on a different (and possibly a much larger) Dictionary. However, you can assume that the file `Dictionary.txt` follows the above format, i.e., each word is displayed after `word:` , its frequency is on the next line after `frequency:` , and its definition is given on the next line after `definition:` followed by an empty line. Also, you can assume that each word in the dictionary consists of only lowercase English letters (e.g., they do not contain white spaces, hyphens or other symbols).

1.2 Output

Once you have created a Trie, your program must ask the user to enter a prefix. It must then display the word with the highest frequency, its definition and the number of words that have the input text as the prefix. If there are more than one words with the highest frequency, you must display the alphabetically smallest word (e.g., if prefix is "be", and "best" and "beast" both have the same frequency (and highest among all words matching the prefix), your program must display "beast" because it is alphabetically smaller than "best"). The program must keep asking the user to enter another prefix and terminate only when the user enters *** instead of the prefix. Below is a sample execution of the program for the small data shown in the **Input** section above.

```
Enter a prefix: adv
Auto-complete suggestion: advisory
Definition: Having power to advise; containing advice; as, an advisory
council; their opinion is merely advisory.
There are 3 words in the dictionary that have "adv" as a prefix.

Enter a prefix: adve
Auto-complete suggestion: adventure
Definition: To try the chance; to take the risk.
There are 2 words in the dictionary that have "adve" as a prefix.

Enter a prefix:
Auto-complete suggestion: advisory
Definition: Having power to advise; containing advice; as, an advisory
council; their opinion is merely advisory.
There are 4 words in the dictionary that have "" as a prefix.

Enter a prefix: bat
There is no word in the dictionary that has "bat" as a prefix.

Enter a prefix: ***
Bye Alice!
```

Note that the third prefix entered by the user in the above example is an empty string in which case the most frequent word in the whole dictionary is returned.

1.3 Complexity/implementation requirement

Your program will have two components. First, you will construct a Trie. Then, for each of the user's entered prefix, you will need to return the relevant results. The worst-case time complexity to construct the Trie must be $O(T)$ where T is the total number of characters in `Dictionary.txt`. The worst-case space complexity of your algorithm must also be $O(T)$. The complexity requirements to return results for each of the user's entered prefix is $O(M + N)$ where M is the length of the prefix entered by the user and N is the total number of characters in the the word with the highest frequency and its definition. Note that this is optimal because the size of the output string is $O(M + N)$ and no algorithm can achieve a better complexity. Note that N may be equal to M (i.e., the word is the same as the prefix and the definition

is empty). Also, note that, in this assignment, we are assuming that each string comparison takes $O(M)$ where M is the number of characters.

Important: You must implement your own Trie and are not allowed to use publicly available implementations. Also, you are **NOT ALLOWED** to use Python dictionary (i.e., hash tables) or linked lists for implementing Trie. You must use arrays (called lists in Python) to implement the nodes of the Trie (as explained in the lecture slides).