FIT2004 S1/2019: Assignment 3 questions

DEADLINE: Friday 10th May 2019 23:55:00 AEST
LATE SUBMISSION PENALTY: 20% penalty per day. Submitting after 5 days of deadline, you will get 0. Even more, if you are late less than a day, you will also be penalized accordingly. For special consideration, please complete and send the *in-semester special consideration* form with appropriate supporting document **before deadline** to `fit2004.allcampuses-x@monash.edu`.
PROGRAMMING CRITERIA: It is required that you implement this exercise strictly using **Python programming language** (version should not be earlier than 3.5). This practical work will be marked on the time, space complexity, coding standard and functionality of your program and your report analysis. If you do not follow the coding standard or program design mentioned in the specification, we will be penalized 20% of your total marks.
SUBMISSION REQUIREMENT: You will submit a zipped file (should be named as `studentId_A3.zip`, e.g. if your student id is XXXX, the name of zipped file must be `XXXX_A3.zip`. Moreover, your zipped file should be ONLY in `.zip` extension) containing your Python program file (named `trie.py`) as well as a PDF file (named `Report_A3.pdf`). The PDF file must give the detail of your solution along with the **worst-case** space and time complexity. Penalties will be applied if you fail to submit the PDF file. The zipped file is to be submitted on Moodle before the deadline. Draft submission will be considered as not-submitted and will not be marked. Also make sure that your zipped file contains only `trie.py` and `Report_A3.pdf`.
PLAGIARISM: The assignments will be checked for plagiarism using an advanced plagiarism detector. 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 ACCEPTED. Please do not share your solutions partially or/and completely to others. If someone asks you for help, ask them to visit us during consultation hours for help.

In assignment-3, you need to solve following two tasks.

Task 1: Querying a database

Input

Input is a text file, named `database.txt`, containing N records: *record index*, *identification number*, *first name*, *last name*, *phone number*, and *email address*. In the input file, every line is an entry that represents the details of one person only. No two lines are identical, though some fields may be the same (e.g. same first name). The indices start at 0 and increase by 1 each line, so they are unique. The identification numbers are also all unique. Moreover, each piece of information of a person is *space-separated*. Below is the example of the part of the large text database `database.txt`

```
0 20876514 David Williams 0398764532 davwil@gmail.com
1 20876515 David Miller 0423567854 david1967@yahoo.com
2 20876526 Jonathan Squire 0399762314 jonie0107@bigpond.com.au
3 20876538 Zhongxian Shao 0459763457 shao1984@gmail.com
4 20876517 Davis Williamson 0496774832 dav67@gmail.com
```

Format of each line in the input:

```
Record_index Identification_no First_name Last_name Phone_number Email_address
```

In the input file, record indices, identification numbers and phone numbers are only **integer**. Identification numbers can have different length. First name and last name contains only English alphabets in both cases. Email address is the combination of alpha-numeric values with 4 special characters: `at(@)`, `dot(.)`, `hyphen(-)` and `underscore(_)`.
As mentioned earlier, each line contains only one person's information, therefore, the number of lines in the input file is N. The maximum length of a single record is M characters, so the size of the input file `database.txt` will be O(NM).

Functionality

In this task you will solve the following problem: given a file of records and a query which consists of two parameters, `id_prefix` and `last_name_prefix` to find the indices of all records which have an identification number (id) whose prefix is `id_prefix` and a last name whose prefix is `last_name_prefix`.
To solve this problem, write a function `query(filename, id_prefix, last_name_prefix)`. This function finds all records whose identification numbers start with `id_prefix` and whose last names start with `last_name_prefix` and returns a list of their indices (which are given in the first column of the input file). This list can be empty. This list does not have to be sorted, it can be in any order.

Complexity requirement

In order to do this efficiently, you will first construct appropriate *TRIEs* inside this function. The construction of these tries should take, in total, $O(T)$ times where T is the number of characters in all identification numbers *and* all last names ((note that it does not include the time complexity needed to read the input file in O(NM)). The space complexity should be $O(T + NM)$. The queries should take $O(k + l + n_k + n_l)$ times, where k is the length of `id_prefix`, l is the length of `last_name_prefix`, n_k is the number of records matching the `id_prefix` and n_l the number of records matching the `last_name_prefix`.

Task 2: Reverse substrings search

In this task, you will be given a long string of text. You need to find all substrings of length $>$ 1 whose reverse also exists in the text (not necessarily at the same position).

For example, if the original text is “cabcdbadccc” then the output should be [[“ab”,1], [“ba”,5], [“cd”,3], [“dc”,7], [“ccc”,8], [“cc”,8], [“cc”,9]]. Notice that "Palindromic" substrings are a special case (eg “ccc”), where the string and its reverse exist at the same position.

You need to write a function `reverseSubstrings(filename)` which takes as input a filename where the file contains a single line containing only lowercase a-z characters. The function returns a lists of lists, where each inner list will contain two values. The first value will be a substring with length >1 whose reverse exists in the string, and the second value will be the index of that substring in the input text. There is no order requirement for the output list, as long as it contains all the correct values.

Complexity requirement:

The function should run in $O(K^2 + P)$, where K is the total number of characters in the input string and P is the total length of all substrings whose reverse appears in the string. The space complexity should be $O(K^2 + P)$.

Output

The output/display of your program will be as below:

```
TASK-1:
-----------------------------------------------------------------------
Enter the file name of the query database : database.txt
Enter the prefix of the identification number: 2087651
Enter the prefix of the last name : Wil
-----------------------------------------------------------------------
2 record found
Index number : 0
Index number : 4
-----------------------------------------------------------------------
TASK-2:
Enter the file name for searching reverse substring: string.txt
-----------------------------------------------------------------------
ab(1), ba(5), cd(3), dc(7), ccc(8), cc(8), cc(9)
-----------------------------------------------------------------------
Program end
```

None of the above mentioned functions: `query` and `reverseSubstrings` will take input from user or give output to user. Therefore, you will use `if __name__ == ‘‘__main__’’` module to take user input, to gives output and to call the above discussed functions to get desired result. Keep in mind, this module should be written inside `trie.py` file.

Comments and docstring in the code

You should use following docstring format for ALL FUNCTIONS in your program:

```
def function name([parameters]):
    ‘‘‘
```

```
    Functionality of the function
    Time complexity: Best:
                     Worst:
    Space complexity: Best:
                      Worst:
    Error handle:
    Return:
    Parameter:
    Precondition:
    '''
```

You can add line comments where required.

Crash prevention

It is the part of the assignment to make your program safe from any kind of crash/ interruption. It is good to use all possible kind of exception handlers to save your program from being crashed/ interrupted. If your program crashed/ interrupted, the examiner/ assignment marker is not obliged to resolve the issue and/ or to mark the assignment. In this case, you will receive **zero**.

Use of in-built functions and data structures

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).

```
-=o0o=-
  END
-=o0o=-
```