Project
(CS-302: Design and Analysis of Algorithms, Fall-2020)

Due Date and Time: December 22, 2020 (1230 hrs) **Weight: 15%**

- *Late submission is not allowed.*
- *Marks will be deducted for not following the file/folder naming instructions (for your submitted files and/or folders).*
- *The project has to be done* ***individually***. *Cheating and copying of any sort will not be tolerated. Help from any external person (a person who is not taking this course) if found may either result in zero marks in the project or F-grade in the course. To abstain from unavoidable circumstances at the end and having panic you are advised to start working on the project from the first day.*

Implement a program in C++, which prints out all anagrams of a specified string. Two strings are anagrams of one another if by rearranging letters of one of the strings you can obtain another. For example, the string "toxic" is an anagram of "ioxct". For the purpose of this assignment, we are only interested in those anagrams of a given string which appear in the dictionary. The dictionary you should use has been provided with the assignment as "words.txt".

Algorithm and Implementation

Since, we will be performing multiple anagram queries, our first step is to load all of the (25,000) words in the dictionary into an appropriate data structure. A primary requirement is that one must be able to efficiently search this data structure to look for anagrams of a given word. A clever trick that we will use to facilitate this is to first sort the letters of every word we insert into our data structure (you may use any sort you wish to produce a key for each word. For example, the key for the string "toxic" is "ciotx", similarly the key for both "star" and "rats" is "arst". We will then use a hash table to store pairs of strings, where the pair consists of the original word and its key.

When performing insertions into the hash table, we will compute the hash of the key of the word to compute the correct bucket (location in the hash table). This approach guarantees that all words which are anagrams of one another are stored in the same bucket of the hash table. Similarly, when we are searching for anagrams, we will first compute the key of the word we are searching for, then hash the key, then search that bucket for anagram matches. You should feel free to use any method for appropriate hash function for hashing strings (but please cite any source which you use in the project report). Also, make sure your function is efficient and does not hash completely unrelated sets of anagrams to the same bucket *if possible*. If it does, handle the collisions as you see fit (e.g. linked processing). Also note that if you must probe for a given set of anagrams in time greater than or equal to O(lgn), then you must revise your hash function. You will be graded heavily on the performance of the efficiency of your function.

Details

The hash table code which you provide only needs to have the minimum functionality needed to solve this problem. You may fix a size for your hash table for efficient searching. It is recommended that the final hash table you submit contain at least 25,000 buckets. (For debugging your code, it is suggested that you work with a much smaller practice dictionary, perhaps 10 words, and a much smaller hash table, perhaps 8-10 buckets (depending on whether or not there are any anagrams in the dictionary).

Make sure your table size is prime to help reduce collisions. Remember, it is ok to sacrifice space for speed -- that is what hashing is all about. That said, your table should not be bigger than 200,000.

You may disregard any words in the dictionary which contain any punctuation characters. Also, you should convert any uppercase characters to lowercase (thus you are only representing words that contain all lower case characters).

Your program should read anagram queries from an input file ("input.txt"). Each query in the file will be on its own line and will simply consist of a string. The output file ("output.txt") should contain the original string, then the number of matching anagrams, followed by those anagrams. An example input file and the resulting output file have been provided. Your output file should match this format exactly, except that the matching anagrams you output may be ordered differently.

Do not count a word as an anagram of itself. Do not use any built-in function or template for hash function or hash table.

Submission instructions

All submission steps must be done by the deadline. Late submission (of any kind) will not be accepted.

- Properly format (table of contexts, numbering of sections /subsections etc.) your project report. It should include different algorithms and data structures that you have used in your project along with their complexity analyses. It should also include your test cases and the result of the test cases. Do not add your programming code in the report. Use "Times New Roman" font with font size 11. Use paragraph spacing of 1.5 lines.

Submission on SLATE

- Submit your project source files as a single zipped file having the format: [roll_number]-Project_code.
- Submit the soft copy of your project report without zipping using the file name format: [roll_number]-Project_report.
- Before submission, make sure that you are submitting the latest version of your project code and report. Excuses like submission of a wrong/early version shall not be entertained.

Submission by e-mail to kashif.munir@nu.edu.pk

- E-mail your zipped project (**source codes without executable files**) and zipped project report to **kashif.munir@nu.edu.pk. CC the email to yourself for verification of correct attachments.** Follow the naming conventions mentioned above. Note that if your zipped project has an executable file, it will likely be bounced back.
- Subject of the e-mail should be [roll_number]-Algo-Project.

Make sure that you submit the same version of contents at all both places (SLATE and e-mail). We may take demonstration from any of these. Failure to follow the instructions will result in deductions of marks.