CS 211 Data Structures and Algorithms Lab
Autumn, 2022

Assignment no.	7
Objective	To find Anagrams using Hashing
Total marks	10
Due date (without penalty)	10th October (Monday) 11:59 pm
Cut-off date (with penalty - 5%)	13th October(Thursday) 11:59 pm
Penalty for violating naming convention(s)	5%

An **Anagram** is a word formed from another by rearranging its letters. For example 'brainy' is an anagram of 'binary' and vice versa.

Command-line argument:

Your program should receive three command-line arguments: a file containing words, the size of the hash table, and a query file. For example, a typical run of your program can be **./a.out words.txt 25000 query.txt** .

Inputs:

An input file containing words: The first command-line argument is the path of a text file containing English words. The file contains one word per line.

The size of the hash table: The second command-line argument is a positive integer which denotes *how many slots should be there in the hash table you create*.

A file containing queries: The third command-line argument is a text file containing a list of 'words', one per line. Here a word may not have any meaning.

Task:

Let *m* be the size of the hash table (as given by the second command-line argument). Use the hash function *h* described below. Given a string, the hash function sums the ASCII values of the characters in the string and takes the remainder obtained by dividing the sum with *m*.

Example:

h('brainy') = 98+114+97+105+110+121 % *m;* If m=25000
= 645

Your hash table should resolve collisions by chaining. Every slot in the hash table should contain a pointer to a linked list. Take the words one by one from the input file containing words, apply the hash function (as shown in the above example), and insert the word in the linked list associated with the slot given by the hash function. The insertion should always be done at the *beginning of the linked list*.

Take every ‘word’ in the file containing queries: (i) hash it; (ii) list all anagrams of the ‘word’ present in the linked list associated with the slot given by the hash function. *Note that the hash function we defined will hash all anagrams of a word to the same slot*. So, to obtain the anagrams of a word, it is enough to search in the linked list associated with the slot. Also note that there can be other words, which are not anagrams in the same linked list. Your program should be able to ignore them while collecting all anagrams of the word.

Output:

The output of your program should be in a file named ***‘anagrams.txt’*** . Corresponding to every line in the input query file, there should be a line in the output file. As mentioned before, every line in the query file contains a ‘word’, which may or may not be meaningless and may or may not be in the file containing the words. The output line for a word is the list of all anagrams (of the word) present in the file containing words. Note that, if the query word is in the words file, then the word is also listed as an anagram of the word and if the query word is not in the words file, then it is not listed as an anagram of the word. The anagrams of a word are listed in a single line where two anagrams of the same word are separated by a single white space. The anagrams are listed in the reverse order of the order given in the words file. That means if the query word is ‘brainy’, the output line contains ‘brainy’ before ‘binary’ (assuming ‘brainy’ comes after ‘binary’ in the words file). *Note that this requirement is natural as you are asked to insert always at the beginning of the linked list associated with a slot*.

Submission:

- The program you submit should output ***anagrams.txt*** when run.
- The main file of your program should be named as <roll no>.<extension>, where roll no. specifies your roll no. and the extension depends on the language you choose (Usage of C is mandatory for this assignment). Ex: 210010001.c.
- Do the stress test of your program well before submission.
  (i) You may use the attached sample input files for testing; the corresponding output files are also attached;
  (ii) We have some hidden inputs with us to test your program. *The mark you obtain is purely based on whether your program correctly gives outputs for the hidden inputs*.
- If your program has only a single source file, please submit the file as it is. If your program has multiple source files, please submit your code as a zip file where the name of the zip file should be your roll number. It is important that you follow the ***input/output conventions*** exactly (including the naming scheme) as we may be doing an automated evaluation. ***There will be a penalty of 5% (on the mark you deserve otherwise) if you do not follow the naming conventions exactly.***
- Follow some coding style uniformly. Provide proper comments in your code.

- *Submit only through moodle*. ***Submit well in advance***. Any hiccups in the moodle at the last minute is never acceptable as an excuse for late submission. Submissions through email or any other means will be ignored.
- Acknowledge the people (other than the instructor and TA) who helped you to solve this assignment. The details of the help you received and the names of the people who helped you (including internet sources, if applicable) should come in the beginning of the main file as a comment. ***Copying others' programs is a serious offense and a deserving penalty will be imposed if found.***

Evaluation:

- To consider for the first evaluation without penalty, you have to submit your program by the due date. If you submit after the ***due date*** but on or before the ***cut-off date***, ***there will be a penalty of 5% on the marks you deserve otherwise***.
- If you do not submit by the ***cut-off date,*** your program will not be considered for the first evaluation.
- We will do the first evaluation after the cut-off date. *The marks you obtain will be proportional to the number of correct lines in the output files*. We will use the ***'diff'*** program to check the differences between the correct output file and the output file generated by your program. So, you may verify the correctness of the output file by using the diff program with a sample output file before submission. (See the man page of diff for more info).
- We will do the second evaluation later. This is for those who want to improve their marks obtained in the first evaluation or who do not submit for the first evaluation. There will be ***a penalty of 20%*** for those who are submitting for the second evaluation. The details of the second evaluation will be shared later.