PROBLEM STATEMENT: ANAGRAM GENERATOR

Goal: The goal of this assignment is to get some practice with collision resolution and hash functions. On the side you will also learn basic string manipulations. It's a fun assignment where the task is to find valid anagrams of a given input.

Problem Statement: You are given a vocabulary V of (lowercase) English words. It uses letters of English alphabet [a-z], digits [0-9], and the apostrophe symbol [']. No other characters are used in the vocabulary V. Your goal is to print out all valid *anagrams* of an input string. The input string will be a sequence of at most 12 characters.

Anagram: Two strings are anagrams of each other if by rearranging letters of one string you can obtain the other. For example, "a bit" is an anagram of "bait", and "super" is an anagram of "purse". Note that we can add spaces at will, i.e., we won't count spaces when matching characters for checking anagrams.

In this assignment, you will load V from the text file and then be ready to compute anagrams. You will be provided an input file also in the text format. In both vocabulary and input files there will be one string written per line. Your goal will be compute all valid anagrams (i.e., each word within your anagram must be present in V) of all input strings. After computing all valid anagrams of one string you must output a '-1' to indicate that you are done computing anagrams of this string. For the purpose of this assignment, you only have to compute anagrams with a maximum of 2 spaces in them (i.e., three words at most). However, each permutation of these words will also be a valid anagram.

This is the first assignment in the course where you will be evaluated not only on the *correctness* and *complexity* of your code, but also on the *runtime efficiency* of the code. You can compute the time taken for your code to run using the built-in getTimeMillis() command.

Vocabulary File: The vocabulary file (vocabulary.txt) will be provided in the resources of the assignment. The first line of the Vocabulary will indicate the number of words in the Vocabulary (V), followed by one word per line (all lowercase and no spaces). A sample vocabulary.txt is given below:

```
6

a

it

bit

bat

tab

i
```

Input File: The input file (input.txt) will be an input to the code at runtime. The first line will have the number (K) of input strings. This will be followed by K lines, with one string per line. It will have only lowercase letters, digits, and apostrophe. It will not have a space. A sample input.txt is given as under

```
2

bait
```

```
bb
```

Output File: You will produce all valid anagrams of each input string and output -1 after finishing with one input and moving onto the next. The output for a particular string should be in **lexicographic order**. Lexicographic ordering is done based on ASCII codes: i.e., lowercase>digits>apostrophe>space. For example, for the input file above you will output:

```
a bit

bat i

bit a

i bat

i tab

tab i

-1

-1
```

Note that for the second input word there were no valid anagrams found. Also note that the number of '-1's in the output should be exactly same as the number of input words in input.txt. Your output must be produced on stdout (without any other extra information).

Further note that output anagrams should not have contiguous spaces. They should not start with a space, or end with a space. These will be required for correctly autograding your assignment.

Hashing: The main purpose of the assignment is to have you store vocabulary appropriately and have you check for anagrams efficiently. There may be many ways to store the vocabulary, but in this assignment you must hash each valid word. You will have to implement your own hash function and your own collision resolution. You may use chaining, or open addressing with any probe sequence, as you see fit. The goal is that your anagram computation should be as efficient as possible. You may use any function within Java built-in String class, except hashCode() or any other inbuilt hash functions.

Tip: To compute better time efficiency not only will you have to implement a good hashing mechanism, you will also have to create an optimized approach to search through the space of anagrams. This may take some trial and error, so start early!

For Humor: You must find some friend of yours (either in the class or otherwise) and output a funny anagram of their name. Share this anagram with TA at the time of demo. There are no points for a more humorous anagram, although there are points for completing this part of the task.

Code: Your code will be run using the following command:

```
javac Anagram.java

java Anagram vocabulary.txt input.txt
```

This implies that we can change the vocabulary.txt at the time of final evaluation. However, its size will be in the range of the size of the vocabulary.txt we are providing with the assignment. Also, there will be no words in the vocabulary that have sizes 1 or 2. That is, all valid words will be at least three characters long.

What is being provided?

The folder contains the following files:

vocabulary.txt

input.txt (sample test case)

What is allowed? What is not?

1. This is an individual assignment.

2. Your code must be your own. You are not to take guidance from any general purpose code or problem specific code meant to solve these or related problems.

3. You are not allowed to use built-in (or anyone else's) implementations of hash functions or hashing scheme. A key aspect of the course is to have you learn how to implement hashing.

4. You are allowed to use built-in Java String functions. You are also allowed to use built-in sorting functions.

5. You should develop your algorithm using your own efforts. You should not Google search for direct solutions to this assignment. However, you are welcome to Google search for generic Java-related syntax.

6. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class.** Please read academic integrity guidelines on the course home page and follow them carefully.

7. Your submitted code will be automatically evaluated against another set of benchmark problems. You get significant penalty if your output is not automatically parsable and does not follow input-guidelines.

8. We will run plagiarism detection software. Anyone found guilty will be awarded a suitable penalty as per IIT rules.