Isomorphic String Analysis

Often, we are asked to create code capable of pattern matching, analysis, and/or generation. This can be used to predict activity, detect fraudulent or artificial data, or even assist in areas of cryptography. Let’s gain some experience in pattern analysis using the concept of isomorphic strings.

**Requirements**

Two or more strings are considered isomorphic if they have equivalent character frequency patterns. For example, the words “egg”, “add”, “gag”, and “bee” are all considered isomorphic. They share the same (or relatively equivalent) character frequency patterns (one character occurs once, the other twice). This kind of isomorphism is sometimes referred to as “plain” or “loose”. In loose isomorphism, the ordering of the letters is not important, only the occurrence frequencies matter.

Two or more strings are considered exactly isomorphic when their character frequency patterns are exactly equal. For example, “egg”, “add”, and “bee” are all exact isomorphs, but “gag” is not. While “gag” is a loose isomorph of this word group, it is not an exact isomorph.

At its core, isomorphic analysis is case-sensitive. “Gag” is not an isomorph (neither loose nor exact) of “yay”. The individual occurrences “G” and “g” are counted separately as they are not considered the same character.

You will build a simple console app that does the following:

* Allow the user to specify a path to a file. This file will contain a collection of words. Each word will appear on its own line.
* Analyzing the words, you will create three mappings:
* Exactly isomorphic words
* Loosely isomorphic words
* Non-isomorphic words
* Exact isomorphic words will be mapped based on their exact character occurrence pattern. Exact character occurrence patterns list the first index (starting at 0) at which that letter first appeared. If the letter repeats, the original index is repeated. Samples of exact character occurrence patterns follow:
* warrior –0 1 2 2 3 4 2
* egg – 0 1 1
* roar – 0 1 2 0
* suspicious – 0 1 0 2 3 4 3 5 1 0
* Loose isomorphic words will be mapped based on their character frequency pattern, normalized so that the frequencies appear from least to greatest. The frequency is simply a count of how many times that character appeared in the string.  
  Samples of loose character frequency patterns follow:
* warrior – 1 1 1 1 3
* egg – 1 2
* roar – 1 1 2
* suspicious – 1 1 1 2 2 3
* Your application will output these mappings in the following format:  
  Exact Isomorphs  
  0 1 0: gag tot yay  
  0 1 1: add egg foo  
  0 1 1 2: look meet seep seer took  
  0 1 2: ate bar eat fit sap  
    
  Loose Isomorphs  
  1 1 1: ate bar eat fit sap  
  1 1 2: look meet seep seer took  
  1 2: add egg foo gag tot yay  
    
  Non-isomorphs  
  aaa fear mates
* Your output MUST BE EXACT in its formatting based on the above example. Spelling, casing, and spacing will matter. Each map group is labeled (ex. Exact Isomorphs), each subgroup is preceded by the pattern, each value in the pattern is separated by the space character, the pattern ends with a colon (“:”), each string in the subgroup is listed alphabetically (based on standard ASCII values), and each map group is separated from the next with a single empty line. There is no empty line after the Non-isomorphs map group, the output simply ends.
* Your output will be written to a file called “Output.txt” as well as being written to the console.

**Rubric**

**Automatic Zero:** You abuse the main method, your application throws an exception at any point, or your output cannot be tested for any reason.

This lab will be graded as follows:

(5 point) Your solution runs and correctly writes the output.txt file and to the console

(10 points) Exact isomorphs are correctly identified

(10 points) Loose isomorphs are correctly identified

(5 points) Non-isomorphs are correctly identified