Introduction to Computer Science II
Lab 6
Chi Square testing

1 Testing your shuffle

1.1 Basics of the chi-squared test

Recall our discussion of the *chi-squared* statistical test. One of its primary uses is to determine how probable a distribution of values is assuming that it should be *independent* and *uniform*. If, for a large number of values, the distribution is measured by this test to be unlikely, then we must consider that the distribution may not be independent and uniform—it may be skewed in some way.

Recall, for reference, that the chi-squared test is calculated with this formula:

$$\chi^2 = \sum_{i=1}^{n} \frac{(H_i - E)^2}{E}$$

The variables for the formula are defined as follows:

- n is the range of the values.
- E is the *expected* number of instances of each value. That is, if k values are used, then $E = \frac{k}{n}$, distributing those k values uniformly over the n possibilities.
- H is a *histogram* of the number of instances of each value in the range. That is, H_i is the number of times that the value i (which must be in the range from 1 to n) was observed.

As an example, consider the problem of determining whether a shuffling algorithm is randomnly permuting the elements of a list uniformly and independently. You could apply the shuffling algorithm, k times, to lists of n unique values. Each time, you could track where just one of the values ends up in the shuffled result. If the value is found in position p, you increment H_p. You would ultimately expect that the value would be found in each of the n positions $\frac{k}{n}$ times; thus, if the shuffling algorithm is a good one, then each entry in the histogram would have a value near $E = \frac{k}{n}$.

You would then calculate the χ^2 value for this set of values. Knowing that the *degrees of freedom*[1] is $n - 1$, you would then search for a web-page that allows you to calculate the *p-value*—the probability—of obtaining that particular χ^2 value with that number of degrees of freedom.

[1] A statistical term for the *number of possible outcomes*, or something like that.

1.2 Applying the test

Our hope, with a *substitution cipher*, is that there is no correlation between each character in the cleartext alphabet and the charactrer randomly selected to replace it in the ciphertext alphabet. However, given an implementation of a subtitution cipher, how could we know whether it is achieving this independence? That is, how do we know that it's shuffling the ciphertext alphabet well? Since we don't have access to the internal shuffling algorithm, we must somehow test the substitution itself by using the encryption.

Doing so, we might try repeatedly encrypting a single character, each time using a different key. We could then build a histogram that represents how many times each possible ciphertext character is chosen as the substitute for that single cleartext character. Given enough encryptions of this kind, we would obtain a histogram H whose uniform distribution could be tested—how likely is that distribution the product of chance, given $n = 256$ possible characters over k encryptions?

Note that this test may have flaws. That is, as a method for testing the quality of the substitution cipher, it may not measure it as well as we would like. For example, consider what we would expect from this test when used on a *Caesar cipher* using randomly chosen keys. You may need to devise an improved variant of this test.

2 What you must do

As usual, create a new `lab-6` directory. Copy your files from Project-3. Finally. grab some new files to add to that directory:

- **On `remus/romulus`**, perform the following command:
```
$ cp ~sfkaplan/public/COSC-112/lab-6/*.class .
```
- **On your own computer**, download and extract this zip file.

You will have grabbed a collection of three `.class` files: `SubA.class`, `SubB.class`, and `SubC.class`. You will undoubtedly have noticed, by now, that these are **compiled** classes, rather than the usual `.java` source code files. You cannot open these files;[2] you may only run them.

To be more specific, these are three implementations of the now-familiar *substitution cipher*. However, two of the three are poor implementations of it. One uses a lousy shuffling algorithm, swapping an insufficient number of randomly chosen pairs of elements; the other uses a poor random number source that only provides even numbered values. The third of the trio lacks these weaknesses.

Your task: Write a cipher-testing program. It should apply the *chi-squared test* to a cipher, measuring whether that cipher chooses ciphertext characters for each cleartext character in a uniform and independent manner. Using this testing code, determine which of the three given substitution cipher implementations is the good one.

[2]More to the point, if you open them, they will be indecipherable, containing machine-code instructions for the *Java Virtual Machine*.

3 How to submit your work

Don’t! This lab is for your own exploration.