Caesar Crack

Jonathan Lam

August 25, 2018

The general process of doing this question consists of the following steps.

- Read the sample unencrypted text and store into an array
- Read the encrypted text and store into an array
- Decide on the best shift to use
  - This subtask here is the most complex, and is split up into further subtasks
  - Method 1 (failed approach)
    - Find the most frequent letter in both texts
    - Use that as the shift
  - Method 2 (Using sum of square errors)
    - Calculate frequencies of letters in sample, and store in `freqSample[]`
    - Calculate frequencies of letters in encoded text, and store in `freqEnc[]`
    - Calculate errors for each shift
    - Choose the shift that gives the minimum error
- Apply the shift to the encrypted text to decrypt it, and print out the result

The first two parts is routine and was done in the previous exercise. The last step of decrypting text given a known shift was also done in a previous exercise. It turns out the 'challenge' of this question is to decide on the best shift.

1 My First (Failed) Approach

My first approach involved calculating the most frequent letter in the two texts, and choosing the shift such that the most frequent letters align. This worked for some (I think only one) of the autotest cases, but not all of them.

2 Method 2

So looking for one letter wasn't good enough, but how about looking at every single letter? This approach calculates the frequencies of the letters (a ratio between 0 and 1) in both sample texts, and stores them in separate arrays.

We then calculate the RSS term.

Let $\{x_i\}$ be the sequence of frequencies for the letters in the unencrypted text ($0 \leq x_i \leq 1$ and $0 \leq i \leq 25$). e.g. x_0 is the frequency for the amount of A's in the text, and x_{25} is the frequency of the amount of Z's in the text.

We similarly define $\{y_i\}$ for the frequencies in the encrypted text.

Let $f(x)$ be the total error value for a shift by x letters. e.g. $f(0)$ represents the error for no shift, $f(3)$ represents the shift from $A \to D$ etc.

Then

$$f(0) = \sum_{i=0}^{25} (x_i - y_i)^2$$

The reason we square it is to ensure the terms stay positive. But why not just use the absolute value? The explanation for this is a bit more complicated and requires knowledge on statistics (I do Actuarial Studies so you can ask me about this).

We can generalise this to get

$$f(x) = \sum_{i=0}^{25} (x_{i+x} - y_i)^2$$

I calculated all the values $f(0), f(1), \ldots, f(25)$ using a loop and stored them in `rss[]`.

The shift was the index of the smallest error term in `rss`.

Before this part, you should have already written functions and whatever to calculate arrays `freqSample` and `freqEnc` which are the frequencies for the Sample and Encoded text, respectively. This is a simple modification of the code from the previous exercise. Then you should be able to just use this:

```
double rss[26] = {0};
  for (int i=0; i<26; i++) {
    // for each possible shift, calculate RSS
    double total_error = 0;
    for (int k=0; k<26; k++) {
      double error = (freqEnc[(k+i)%26] - freqSample[k]) *
                     (freqEnc[(k+i)%26] - freqSample[k]);
      total_error += error; // hence total_error = sigma (x - x)^2
    }
    rss[i] = total_error;
  }
```

Then the shift is determined using

```
int shift = -min_index(rss, 26);
```

I wrote my own function called `min_index()` which takes an array and size as its arguments, and returns the index of the minimum in the array. For example, if the letter h had a minimum value, the function would return 7 (since h is the 8th letter, but arrays start at 0)

3 Print out Answer

I should have used putchar but too late now

```
i=0;
while (encText[i] != '\0') {
  int ch = encText[i];
  printf("%c", encrypt(ch, shift));
  ++i;
}
```