CS 314 Principles of Programming Languages

A Caesar's Cipher Code Breaker

Due date: Wednesday, November 29, 11:59pm

In this project, you will implement a function that takes as input a sequence of words encoded via an unknown Caesar's Cipher, and returns a function that decodes words in that cipher back into plain text. You then use this function to write a code-breaker that decodes an entire document back into its plain text.

Caesar's Cipher

The cipher translates each letter in a word independently by "shifting" the letter up or down the alphabet by a fixed distance. We assume that we are using the English alphabet with 26 letters in their "usual" order, all lower case: a, b, c, ... z. The function ltv ("letter-to-value") maps each character (letter) to its value, and the function vtl ("value-to-letter") maps each value to its corresponding character (letter). For example, ltv(c) = 2 and vtl(25) = z.

The encryption function for a single letter has the following form:

$$Encript_n(x) = vtl((ltv(x) + n) mod 26)$$

where "n" ≥ 0 represents the amount by which the letters are "shifted" up the alphabet. Note that this "shift" operation "wraps around", i.e., shifting a "z" by n=1 results in an "a". For this project, we assume the values of "n" to be in the range of 0 and 25, i.e., $0 \leq n \leq 25$.

As part of the project, you will first need to implement the encoding function `encode-n` which takes as input the shift value n and returns a function that takes as input a word w, and returns its encoded version:

```
(define encode-n
  (lambda (n)
     (lambda (w)
        ...
)))
```

For this project, **words** are represented as lists of lower case symbols, e.g., the word “class” is represented as ’(c l a s s).

Project Description

Paragraphs are lists of words, e.g., ’((c l a s s) (i s) (a) (l o t) (o f) (f u n)). A **document** is a list of paragraphs.

You are asked to implement two different methods to break the cipher, (1) a brute force version that uses a spell checker, and (2) a version that uses knowedge about the distribution of particular letters in the English language (frequency analysis). Both methods try to find the value of n' such that $(n + n') = 26$.

You will need to write two functions `Gen-Decoder-A` and `Gen-Decoder-B` that take as input a paragraph of encoded words, and return as output a *decode function* that takes as input an encoded word, and returns the plain text of the decoded word. This function can then be used to decode the entire encoded document which may consist of multiple paragraphs. The function `Code-Breaker` takes an encoded document and a decoding function as input, and returns the entire document in plain text.

The two decoder functions implement different strategies to break the encrypted code. Both have advantages and disadvantages. Your scheme program will provide an infrastructure to assess the effectiveness of these decoding approaches for different document types.

Brute Force with Spell Checker `Gen-Decoder-A`

The algorithm for the brute force code breaker is rather simple. The input paragraph of encoded words using the unknown n are further encoded for each possible value of n'. For each n' value, a spell checker determines whether the resulting words are words in the English language. The value of n' for which most words are spelled correctly is assumed to be decoding value.

For this method, you will need to implement a spell checker. You can implement your spell checker using the dictionary of words in file `dictionary.ss`. You will need to implemented the spell checker `spell-checker` that takes as input a word, and returns the truth value #t or #f. A brute force version of the spell checker may just see whether the word occurs in the dictionary or not (set membership).

Frequency Analysis `Gen-Decoder-B`

This code breaker is based on the fact, that in the English language some letters occur more often than others. Knowing the distribution of the different letters, this method just counts the number of occurrences of each letter in the encoded words. If there are enough words to be statistically relevant, the letter distribution can be used to identify the most common letters in English, namely 'e', 't', and 'a', in that order. In other words, the assumption is that the most frequent encoded letter has to correspond to the letter 'e', etc. This means that the shift between the encoded letter and 'e' has to be the value n. The same n has to work for the other frequent letters as well. From that, we can determine the decoding value n'. Note that this method needs a large enough set of words in order to be successful. In other words, it may not work well for very short paragraphs.

Implementation

For the implementation of these functions, you can use standard built-in Scheme functions such as `append`. You should use the `reduce` function at least once in your implementation. The definition of `reduce` is given in file `include.ss`. **You must not use assignment (set!) in Scheme**.

How To Get Started

Copy the 5 files `decode.ss`, `include.ss`, `test-dictionary.ss`, `dictionary.ss` and `document.ss` into your own subdirectory. You will implement your code breaker in file `decode.ss`. File `test-dictionary.ss` contains a small dictionary consisting of only six words. Use it to debug your program. File `dictionary.ss` contains a list of over 45,000 words allowing you to generate a realistic spell checker. You must not change this file. Finally, file `document.ss` contains two sample documents to test your encoder, decoder, etc. . You should use your own test cases as well.

Grading

You will submit your version of file `decode.ss` via Sakai. More detailed instructions will be posted later.

Your programs will be graded based on programming style and functionality. Functionality will be verified through automatic testing on a set of test cases.

Questions

All questions regarding this project should be posted on our Sakai web site. **START EARLY** since you will run into issues that will need to be resolved on the way. This is a feature of the project, not a bug!