COP 3503 Recitation – Spring 2015

The aim of this recitation is to write a Java program to crack the *Caesar cipher* using simple frequency analysis.

Submit any necessary files to the assignment as a .zip file webcourses.

Introduction

A *cipher* is a method for encoding a message by replacing each character of the message by another character. One of the simplest examples is the *Caesar cipher*, which is said to have been used for military purposes by Julius Caesar. To encode a message, Caesar simply replaced each letter in the message by the letter three places further down the alphabet, wrapping around at the end of the alphabet. For example, the message

haskell is fun

would be encoded as

kdvnhoo lv ixq

More generally, the shift factor of three used by Caesar can be replaced by any natural number between one and twenty-five, thereby giving twenty-five different ways to encode a message. For example, with a shift factor of ten, the original message above would be encoded as

rkcuovv sc pex

The aim of this assignment is to write a java program that can automatically decode such messages, by using letter frequencies to determine the most likely shift factor that was used to encode the message.

Encoding and Decoding

- **Exercise:** Define a method

```
int let2nat( char c );
```

that converts a lower-case letter in the range 'a' to 'z' into the corresponding natural number in the range 0 to 25. For example:

```
let2nat( 'a' ) -> 0
let2nat( 'z') -> 25
```

- **Exercise:** Define a method

```
char nat2let(int code)
```

that performs the inverse method to *let2nat*. For example:

```
nat2let(0) -> 'a'
nat2let(25) -> 'z'
```

- **Exercise:** Using *let2nat* and *nat2let*, define a method

```
char shift(int shftAmt, char c)
```

that applies a shift factor in the range 0 to 25 to a lower-case letter in the range 'a' to 'z'. Characters outside this range, such as upper-case letters and punctuation, should be returned unshifted. Take care to ensure that your method wraps around at the end of the alphabet. For example:

```
shift( 3, 'h')  -> 'k'
shift (3, 'z')  -> 'c'
shift (3 'H')  -> 'H'
```

- **Exercise:**
  Using *shift*, define a method

  ```
  String encode(int shftAmt, String str)
  ```

that encodes a string using a given shift factor. For example:

```
encode(3, "haskellisfun") -> "kdvnhoolvixq"
```

- **Exercise:** Define a method

```
String decode(int shftAmt, String str)
```

that performs the inverse method to *encode*. For example:

```
decode(3, "kdvnhoolvixq") -> "haskellisfun"
```

Frequency Analysis

In English text, some letters are used more frequently than others. By analysing a large volume of text, one can derive the following table of approximate percentage frequencies of the twenty-six letters of the alphabet:

static double[] table = {8.2, 1.5, 2.8, 4.3, 12.7, 2.2, 2.0, 6.1, 7.0, 0.2, 0.8, 4.0, 2.4, 6.7, 7.5, 1.9, 0.1, 6.0, 6.3, 9.1, 2.8, 1.0, 2.4, 0.2, 2.0, 0.1}

For example, this table shows that 'e' occurs most often, with a frequency of 12.7 percent, while 'q' and 'z' occur least often, with a frequency of 0.1 percent. You will need to copy this table into your java class.

- **Exercise:** Define a method

```
int lowers(String str)
```

calculates the number of lower-case letters in a string. For example:

```
lowers("haskellisfun") -> 12
```

Define a method

```
int count(char c, String str)
```

that calculates the number of a given character in a string. For example:

```
count('s', "haskellisfun") -> 2
```

- **Exercise:** Define a method

```
double percent(int num1, int num2)
```

that calculates the percentage of one integer with respect to another, returning the result as a floating-point number. For example:

```
percent(2, 12) -> 16.6667
```

- **Exercise:** Using *lowers*, *count* and *percent*, define a method

```
double[] freqs(String str)
```

that returns the list of percentage frequencies of each of the lower-case letters 'a' to 'z' in a string of characters. For example:

```
freqs("haskellisfun") ->
{8.33333,0.0,0.0,0.0,8.33333,8.33333,0.0,8.33333,
8.33333,0.0,8.33333,16.6667,0.0,8.33333,0.0,0.0,
0.0,0.0,16.6667,0.0,8.33333,0.0,0.0,0.0,0.0,0.0}
```

- **Exercise:** Define a method

```
double[] rotate(int n, double[] list)
```

that rotates a list *n* places to the left, wrapping around at the start of the list, and assuming *n* is in the range zero to the length of the list. For example:

```
rotate(1, freqs("haskellisfun") ) ->
{0.0,0.0,0.0,8.33333,8.33333,0.0,8.33333,8.33333,0.0,8.33333,16.6667,0
.0,8.33333,0.0,0.0,0.0,0.0,16.6667,0.0,8.33333,0.0,0.0,0.0,0.0,0.0,8.33333
}
```

- **Exercise:** Define a method

```
double chisqr(double[] os)
```

that calculates the *chi square* statistic for a list of observed frequencies *os* with respect to a list of expected frequencies *es*, which is given by

$$chisqr\ os\ es \quad = \quad \sum_{i=0}^{n-1} \frac{(os_i - es_i)^2}{es_i}$$

where *n* is the length of the two lists *os* and *es*, and xs_i denotes the *i*th element of a list *xs*, counting from zero. For example:

```
chisqr(freqs("haskellisfun")) -> table
202.616
```

- **Exercise:** Define a method

```
int position(double a, double[] list)
```

that returns the first position (counting from zero) at which a value occurs in a list, assuming that it occurs at least once. For example:

```
position(5, [1,3,5,7,11]) -> 2
```

- **Exercise:** Using the methods defined above, define a method

```
String crack(String str)
```

that attempts to decode a string by first calculating the letter frequencies in the string, then calculating the chi square value of each rotation (in the range zero to twenty-five) of this list with respect to the table of expected frequencies, and finally using the position of the minimum value in this list as the shift factor to decode the original string. For example:

```
crack(encode(3, "haskellisfun")) -> "haskellisfun"
```

Note that this method may not be successful if the message is short or contains an unusual distribution of letters. For example:

```
crack (encode(3, "graham")) -> "nyhoht"
rack(encode(3,"thefiveboxingwizardsjumpquickly"))->
"dropsfolyhsxqgsjkbnctewzaesmuvi"
```

Try out your *crack* method on the following example:

myxqbkdevkdsyxc yx mywzvodsxq dro ohkw!