

# Paradigmas de Programação

Fabício Olivetti de França

26 de Junho de 2018

## Cifra de César

### Exercício

A Cifra de César é um algoritmo criptográfico que consiste em reescrever uma frase substituindo cada letra pela letra  $n$  posições adiante no alfabeto. Por exemplo, para  $n = 3$  temos que:

“adoro haskell”

se transforma em:

“dgrur kdvnho”

### Crie o projeto

```
> stack new cifracesar simple
```

No arquivo *cifracesar.cabal* adicione:

```
executable cifracesar
```

```
hs-source-dirs: src
```

```
main-is: Main.hs
```

```
other-modules: Cesar, Test
```

```
default-language: Haskell2010
```

```
build-depends: base >= 4.7 && < 5, QuickCheck
```

No diretório *src* crie os arquivos *Cesar.hs* e *Test.hs*. O arquivo *Main.hs* deve conter:

```
module Main where
```

```
import Test.QuickCheck
```

```
import Cesar
import Test
```

O arquivo *Main.hs* deve conter:

```
main :: IO ()
main = do
  let code    = encode 3 "adoro haskell"
      dec     = encode (-3) code
      cracked = crack code
  print ("Codificado: " ++ code)
  print ("Decodificado: " ++ dec)
  print ("Crackeado: " ++ cracked)
  quickCheck prop_neg_shift
  quickCheck prop_enc_length
```

O arquivo *Cesar.hs* deve conter:

```
module Cesar where

import Data.Char

-- converte uma letra minuscula para inteiro
let2int :: Char -> Int
let2int c = ord c - ord 'a'

-- converte um inteiro para letra minuscula
int2let :: Int -> Char
int2let n = chr (ord 'a' + n)

table :: [Float]
table = [8.1, 1.5, 2.8, 4.2, 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, 9.0, 2.8, 1.0, 2.4, 0.2, 2.0, 0.1]
```

O arquivo *Test.hs* deve conter:

```
module Test where

import Cesar
```

## Cesar.hs

Complete o arquivo *Cesar.hs* definindo as seguintes funções:

```
-- retorna a n-ésima letra seguinte,
-- evite ultrapassar o limite com `mod` 26
shift :: Int -> Char -> Char
```

```
-- aplica a função shift em cada letra da string
encode :: Int -> String -> String
```

É possível *quebrar* a cifra de César utilizando uma tabela de frequências observadas (já definida em `table`). Para isso basta:

- calcular as frequências de ocorrência de cada letra do alfabeto na mensagem codificada
- comparar rotações dessa tabela de frequência com a tabela de valores esperados (`table`)
- a rotação de menor erro é o nosso `n`

Escreva a seguinte função:

```
crack :: String -> String
crack xs = encode (-factor) xs
  where
    factor = head (positions (minimum chitab) chitab)
    chitab = [chisqr (rotate n table') table
              | n <- [0..25]]
    table' = freqs xs
```

Agora defina as seguintes funções:

```
-- quantidade de letras minúsculas em uma String
lowers :: String -> Int

-- conta a ocorrência de um caracter em uma String
count :: Char -> String -> Int

-- dado um n e m, calcule 100*n/m
percent :: Int -> Int -> Float

-- calcule a porcentagem de cada letra minuscula
-- do alfabeto em uma String
-- a porcentagem é a contagem de ocorrência pelo total
-- de letras minúsculas
freqs :: String -> [Float]

-- Calcule a medida de Chi-Squared de duas
-- tabelas de frequência:
-- Soma (Observado - Esperado)2 / Esperado
chisqr :: [Float] -> [Float] -> Float

-- rotaciona uma tabela em n posições
rotate :: Int -> [a] -> [a]

-- retorna a lista de posições que contém um
```

```
-- elemento x
positions :: Eq a => a -> [a] -> [Int]
```

## Test.hs

Complete o arquivo *Test.hs* com as definições:

```
-- aplicando shift duas vezes, uma com o valor negativo, o caracter deve ser o mesmo
prop_neg_shift :: Int -> Char -> Bool
```

```
-- o tamanho da mensagem codificada deve ser o mesmo da original
prop_enc_length :: Int -> String -> Bool
```

Finalmente teste seu código:

```
> stack setup
> stack build
> stack exec cifracesar
```