

Cifra de Cesar

BCC22 Programação Funcional — Aula Prática 02

Lucília Camarão de Figueiredo

A prática deve ser realizada antes da aula prevista para a discussão da mesma. Verifique a data prevista para a discussão de cada aula prática no calendário do curso disponível em www.dcc.ufmg.br/~lucilia/cursos/func. Traga para a aula de discussão o código que você implementou. A discussão não será eficaz se você não tentar fazer os exercícios previamente.

Você pode trabalhar em conjunto com outros colegas, mas deve entender o que está fazendo, pois não irá contar com a ajuda de colegas durante os exames.

A realização das práticas e presença nas aulas são obrigatórias e valem pontos na disciplina.

1 Cifra de Cesar

Quando falamos de criptografia hoje em dia, usualmente estamos falando de codificação e decodificação de mensagens digitais. Entretanto, a codificação de mensagens já era usada há muitos e muitos anos. Um dos melhores exemplos de antigo sistema criptográfico é a Cifra de César, cujo nome vem do fato de que se acredita que tenha sido usada pelo imperador romano Julius Caesar. A idéia é simples: tome a mensagem a ser criptografada e substitua cada letra por outra letra que esteja k posições adiante dela no alfabeto, para um determinado *offset* k (entre 0 e 26, supondo que o alfabeto tem 26 letras). Por exemplo, criptografar a sentença “ESTE É UM GRANDE SEGREDO”, usando um offset de 5, daria como resultado “JXYJ J ZR LWFSIJ XJLWJIT”.

Neste exercício você vai implementar a codificação via Cifra de César. Você pode usar funções pré-definidas na biblioteca padrão de Haskell, para implementar sua solução.

2 Codificando a mensagem

Uma codificação caractere a caractere, tal como a da Cifra de César, pode ser representada por uma lista de pares, onde cada par da lista indica como uma letra deve ser codificada. Por exemplo, para uma cifra para as de A a E poderia ser representada pela lista

```
[('A', 'C'), ('B', 'D'), ('C', 'E'), ('D', 'A'), ('E', 'B')]
```

Embora seja possível escolher qualquer letra como código para qualquer outra letra, vamos considerar uma codificação na qual o código para cada letra é a letra obtida deslocando-se um determinado *offset* k para a direita no alfabeto (considerado como uma lista circular). Vamos considerar um alfabeto com 26 letras, incluindo K, W e Y.

Exercícios

1. Podemos rodar uma lista tomando alguns itens da frente da lista e colocando-os no final da mesma. Exemplo:

```
Main> rotate 3 "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
"DEFGHIJKLMNOPQRSTUVWXYZABC"
```

Abra o arquivo `prat01.hs` e complete a função `rotate :: Int -> [Char] -> [Char]`. Sua função deve retornar um erro (usando a função `error` pré-definida no Prelude) se o número passado como argumento for menor que 0 ou se for maior que o comprimento da lista.

2. Observe a função de teste `prop_rotate`.

- (a) O que essa função testa, precisamente?
- (b) Como essa função assegura que `rotate` não produz um erro?

3. Usando a função `rotate` do exercício anterior, escreva a função `makeKey :: Int -> [(Char, Char)]`, que retorna a cifra obtida com o offset dado. Exemplo:

```
Main> makeKey 5
[( 'A', 'F'), ('B', 'G'), ('C', 'H'), ('D', 'I'), ('E', 'J'), ('F', 'K'), ('G', 'L'), ('H', 'M'), ('I', 'N'), ('J', 'O'),
 ('N', 'S'), ('O', 'T'), ('P', 'U'), ('Q', 'V'), ('R', 'W'), ('S', 'X'), ('T', 'Y'), ('U', 'Z'), ('V', 'A'), ('W', 'B'),
```

A cifra deve mostrar como codificar toda letra maiúscula do alfabeto, não contendo letras duplicadas: cada letra deve ocorrer uma única vez como primeiro componente de um par (e também como segundo componente).

4. Escreva a function `lookUp :: Char -> [(Char, Char)] -> Char`, que obtém o código para um caractere dado, conforme a cifra dada. Se o caractere não ocorrer na cifra, o valor retornado deve ser ele próprio. Exemplos:

```
Main> lookUp 'B' [( 'A', 'F'), ('B', 'G'), ('C', 'H')]
'G'
Main> lookUp '9' [( 'A', 'X'), ('B', 'Y'), ('C', 'Z')]
'9'
```

5. Escreva a função `encipher :: Int -> Char -> Char` que codifica um único caractere, usando a cifra obtida com o offset dado. Exemplos:

```
Main> encipher 5 'C'
'H'
Main> encipher 7 'Q'
'X'
```

6. Um texto criptografado convencionalmente é escrito em letras maiúsculas, sem caracteres de pontuação. Escreva a função `normalize :: String -> String`, que converte um string para letras maiúsculas, removendo todos os demais caracteres que não sejam letras ou dígitos (remove espaços também). Exemplo:

```
Main> normalize "July 4th!"
"JULY4TH"
```

7. Escreva a função `encipherStr :: Int -> String -> String`, que normaliza o string dado e o codifica de acordo com a cifra correspondente ao *offset* dado. Exemplo:

```
Main> encipherStr 5 "July 4th!"
"OZQD4YM"
```

3 Decodificando a mensagem

A Cifra de César é um dos códigos mais fáceis de serem “quebrados”. Diferentemente da maioria dos esquemas de codificação usados hoje em dia, a Cifra de César pode ser quebrada usando um simples algoritmo força bruta que tenta todas as possíveis chaves sucessivamente. A Cifra de César é uma cifra de *chave simétrica*, isto é, a chave contém informação suficiente tanto para codificação como para decodificação.

Exercícios

9. Decodificar uma mensagem codificada fica mais fácil se primeiro transformamos a cifra de codificação em uma cifra de decodificação. Escreva a função `reverseKey :: [(Char, Char)] -> [(Char, Char)]`, que troca de posição os componentes dos pares da cifra dada. Exemplo:

```
Main> reverseKey [('A', 'G'), ('B', 'H'), ('C', 'I')]
[('G', 'A'), ('H', 'B'), ('I', 'C')]
```

10. Escreva as funções `decipher :: Int -> Char -> Char` e `decipherStr :: Int -> String -> String`, para decifrar um caractere e uma mensagem, respectivamente, usando a cifra obtida com o *offset* dado. Exemplo:

```
Main> decipherStr 5 "OZQD4YM"
"JULY4TH"
```

4 QuickCheck – dicas

Para testar a função `rotate` foi necessário garantir que a função de teste não geraria erros. A entrada, gerada aleatoriamente pelo `QuickCheck`, devia obedecer certos critérios — você determinou esses critérios no exercício (2).

No teste `prop_rotate` nos certificamos que a entrada era do tipo apropriado modificando adequadamente os valores de entrada. mas essa nem sempre é a melhor solução, e muitas vezes essa solução nem mesmo é possível. Uma maneira mais geral de garantir que a entrada da função satisfaz determinada propriedade é usar uma implicação `'==>'`. A implicação do `QuickCheck` é semelhante à implicação lógica. Ela é tomada como argumentos duas expressões Booleanas, por exemplo `expr1` e `expr2` (o tipo do resultado é chamado `Property`):

```
expr1, expr2 :: Bool
prop_test :: Property
prop_test = expr1 ==> expr2
```

De modo geral, a propriedade descrita acima vale se `expr1` é `False` ou `expr2` é `True`. Entretanto, para assegurar que todos os testes são relevantes, `QuickCheck` ignora o teste se `expr1` for `False`, e apenas conta como sucesso os testes em que tanto `expr1` como `expr2` são `True`:

```
Main> quickCheck (True ==> True)
OK, passed 100 tests.
Main> quickCheck (False ==> True)
Arguments exhausted after 0 tests.
```

Como você pode ver, `QuickCheck` não continua a gerar valores para sempre; se depois de uma certa quantidade de testes `expr1` ainda não for `True`, ele pára, exibindo a mensagem `'arguments exhausted'`.

Exercícios

11. Para ver como sua codificação funciona, escreva um teste `QuickCheck prop_cipher` para verificar que codificando e decodificando um string com a mesma chave resulta na mensagem original — exceto que em letras maiúsculas e sem espaços ou pontuação (isto é, *normalizada*). Use `'==>'` para garantir que o seu teste não gere erros.

5 Quebrando o código

Um tipo de ataque de força bruta sobre uma mensagem criptografada consiste em decodificá-la usando cada possível chave e então pesquisar no texto resultante pela ocorrência de seqüências de letras comuns na língua na qual a mensagem foi escrita. Se tais seqüências ocorrerem, então a chave é uma candidata a ser a chave realmente usada na codificação da mensagem. Por exemplo, as palavras "the" e "and" ocorrem com grande freqüência em

textos em Inglês. Em “Adventures of Sherlock Holmes”, as ocorrências de "the" e "and" constituem 1/12 de todas as palavras do texto e não existe nenhuma sequência de mais de 150 palavras em que não ocorra "the" ou "and" (veja <http://www.edict.com.hk/textanalyser/wordlists.htm>). A conclusão é que se testarmos uma chave em um texto suficientemente longo e o resultado não contém ocorrências de "the" ou "and", então essa chave pode ser descartada como candidata a ser a chave de codificação da mensagem.

Exercícios

12. Escreva a função `contains :: String -> String -> Bool`, que retorna `True` se o string passado como primeiro argumento ocorre como substring do string passado como segundo argumento, ou retorna `False` em caso contrário. Exemplo:

```
Main> contains "Exampleamp"
True
Main> contains "Examplexml"
False
```

13. Escreva a função `candidates :: String -> [(Int, String)]`, que decodifica o string dado, com cada um das 26 possíveis chaves (*offsets*) e, se o texto decodificado contém "THE" ou "AND", inclui a chave de decodificação e o texto decodificado na lista resultante. Exemplo:

```
Main> candidates "DGGADBCOOCZYMJHZYVMT0JOCZHVS"
[(5,"YBBVYWXJJXUTHECUTQHOJEJXUCQN"),
 (14,"PSSMPNOAAOLKYVTLKHYFAVAOLTHE"),
 (21,"ILLFIGHTTHEDROMEDARYTOTHEMAX")]
```

6 Exercícios opcionais

Como você viu na seção anterior, a Cifra de Cesar não é um método de codificação muito seguro. Nessa seção vamos melhorá-lo um pouco.

Exercícios

14. Escreva a função `splitEachFive :: String -> [String]` que divide o string dado em uma lista de substrings de comprimento 5. Preencha a última parte com o caractere 'X' de modo que tenha comprimento igual ao das demais partes (5).

```
Main> splitEachFive "Secret Message"
["Secre", "t Mes", "sageX"]
```

15. A função pré-definida `transpose` troca linhas e colunas de uma lista de listas:

```
Main> transpose ["123","abc","ABC"]
["1aA","2bB", "3cC"]
Main> transpose ["1","22","333"]
["123","23","3"]
```

Se as linhas de uma lista de listas têm todas o mesmo comprimento, a transposição dessa lista duas vezes resulta na lista original. Use sua função `splitEachFive` para escrever uma propriedade QuickCheck para testar isso. Além disso, mostre um exemplo para ilustrar que isso não necessariamente acontece caso as listas não sejam todas de mesmo comprimento.

16. Escreva a função `encrypt :: Int -> String -> String` que codifica um string usando o seguinte método. Primeiro aplica a cifra de César, depois divide o string resultante em partes de comprimento 5, depois transpõe a lista resultante e junta novamente as partes da lista obtida em um único string.
17. Escreva uma função `decrypt` para decodificar mensagens codificadas segundo o método descrito acima.

7 Apêndice: Algumas funções úteis

Nota: para usar a maioria das funções abaixo você tem que importar `Data.Char` ou `Data.List`.

`ord :: Char -> Int` (Thompson p. 42)

Retorna o código numérico do caractere

Exemplos: `ord 'A' == 65` `ord '1' == 49`

Veja: http://www.zvon.org/other/haskell/Outputchar/ord_f.html

`chr :: Int -> Char`

Retorna o caractere correspondente ao código numérico

Exemplos: `chr 65 == 'A'` `chr 49 == '1'`

Veja: http://www.zvon.org/other/haskell/Outputchar/chr_f.html

`mod :: Int -> Int -> Int`

Retorna o resto da divisão inteira do primeiro argumento pelo segundo

Exemplos: `mod 10 3 == 1` `mod 25 5 == 0`

Veja: http://www.zvon.org/other/haskell/Outputprelude/mod_f.html

`isAlpha :: Char -> Bool`

Retorna `True` se o argumento é um caractere alfabético

Exemplos: `isAlpha '3' == False` `isAlpha 'x' == True`

Veja: http://www.zvon.org/other/haskell/Outputchar/isAlpha_f.html

`isDigit :: Char -> Bool`

Retorna `True` se o argumento é um caractere numérico

Exemplos: `isDigit '3' == True` `isDigit 'x' == False`

Veja: http://www.zvon.org/other/haskell/Outputchar/isDigit_f.html

`isUpper :: Char -> Bool`

Retorna `True` se o argumento é uma letra maiúscula

Exemplos: `isUpper 'x' == False` `isUpper 'X' == True`

Veja: http://www.zvon.org/other/haskell/Outputchar/isUpper_f.html

`isLower :: Char -> Bool`

Retorna `True` se o argumento é uma letra minúscula

Exemplos: `isLower '3' == False` `isLower 'x' == True`

Veja: http://www.zvon.org/other/haskell/Outputchar/isLower_f.html

`toUpper :: Char -> Char`

Se o argumento for um caractere alfabético, converte-o para maiúscula

Exemplos: `toUpper 'x' == 'X'` `toUpper '3' == '3'`

Veja: http://www.zvon.org/other/haskell/Outputchar/toUpper_f.html

`isPrefixOf :: String -> String -> Bool`

Retorna `True` se o primeiro argumento é um prefixo do segundo

Exemplos: `isPrefix "hashaskell" == True` `isPrefix "hashandle" == False`

Veja: http://www.zvon.org/other/haskell/Outputlist/isPrefixOf_f.html

`error :: String -> a`

Sinaliza um erro

Exemplos: `error "Function only defined on positive numbers!"`

Veja: http://www.zvon.org/other/haskell/Outputprelude/error_f.html