Princípios de Programação

Trabalho 4

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Engenharia Informática

2021/2022

O objetivo principal da Criptografia é o de estudar técnicas de enviar mensagens com segredos, que não possam ser compreendidas facilmente por outros que não o recetor desejado. Esta necessidade existiu desde os tempos antigos, por motivos militares, religiosos, políticos ou comerciais.
Neste projeto vamos implementar três técnicas criptográficas — César, Vigenère e de Substituição — para podermos codificar as nossas próprias mensagens, e descodificar as mensagens dos outros!

Considerações prévias sobre mensagens e cifras Para simplificar, podem assumir neste trabalho que as mensagens contêm apenas os seguintes caracteres.

- Letras maíusculas de `A-Z`, que também serão codificadas em letras maiúsculas.
- Letras minúsculas de `a-z`, que também serão codificadas em letras minúsculas.
- Caracteres de pontuação `,;.?!:-()`, que não serão codificados.
- Caracteres brancos (espaços, tabulações, mudanças de linha), que não serão codificados.

Não iremos utilizar quaisquer sinais diacríticos (acentuações, cedilhas, ...).
Iremos utilizar sempre um alfabeto de 26 letras.
Cada um dos métodos a ser implementados requer uma *chave*. Para a cifra César, a chave é um número inteiro. Para as cifras Vigenère e de Substituição, a chave é uma palavra constituída apenas por letras maíusculas.

A. Cifra César Numa cifra César (utilizada frequentemente pelo general e imperador romano Júlio César), cada letra do alfabeto é substituída pela letra n posições à frente, onde n é um número fixo que corresponde à *chave* da cifra. Considerem que o alfabeto é cíclico, isto é, após um `Z` recomeçamos do `A`. Por exemplo para $n = 3$, uma aplicação da cifra de César transforma `A->D`, `B->E`, `C->F` e por aí adiante até `Z->C`.

```
Texto original: As armas e os baroes assinalados
Método: César, chave = 3
Texto cifrado : Dv dupdv h rv edurhv dvvlqdodgrv
```

Para decifrar uma mensagem, sabendo a chave n, basta recuar n casas. Por exemplo para $n = 3$, teríamos `A->X`, `B->Y`, `C->Z`, `D->A` e por aí adiante.

```
Texto cifrado : R pdu vdojdgr, txdqwr gr whx vdo
Método: César, chave = 3
Texto original: O mar salgado, quanto do teu sal
```

B. Cifra Vigenère Uma cifra Vigenère (estudada pelo criptográfo francês Blaise de Vigenère, 1523–1596) é baseada na cifra César, mas em que o desvio de cada letra muda ciclicamente com base na palavra chave.
Suponhamos que conhecemos a palavra chave `HASKELL`. Começamos por substituir cada letra da chave pelo inteiro que lhe corresponde no alfabeto (`A=0`, `B=1`, `C=2`, etc.)

```
Chave: HASKELL
Desvios: 7,0,18,10,4,11,11
```

Isto significa que, para codificar uma mensagem com a chave `HASKELL`, aplicamos um desvio de 7 posições à primeira letra da mensagem, 0 posições à segunda letra, 18 posições à terceira letra, e assim sucessivamente até chegar ao fim da mensagem. Quando chegamos ao fim da chave, retomamos do seu início.
Exemplo:

```
Original: A  s  a  r  m  a  s  e  o  s  b  a  r  o  e  s
Desvios : 7  0 18 10  4 11 11  7  0 18 10  4 11 11  7  0
Cifrada : H  s  s  b  q  l  d  l  o  k  l  e  c  z  l  s
```

O mesmo exemplo, agora sem espaços adicionais.

```
Texto original: As armas e os baroes assinalados
Método: Vigenère, chave = HASKELL
Texto cifrado : Hs sbqld l ok leczls scwtyhlsnsd
```

Notem que os espaços e sinais de pontuação *não são* codificados. A terceira letra da chave (que é `S`, desvio 18) é utilizada para codificar a terceira letra da

mensagem (que é o primeiro 'a' de `armas`) que corresponde ao quarto caracter da mensagem.
Para decifrar uma mensagem, conhecendo a chave, basta repetir os passos de codificação, mas desta vez recuamos (em vez de avançarmos) um número de posições correspondente ao desvio.

```
Texto cifrado : V msb wlwnavy, uflutg ns epb ssv
Método: Vigenère, chave = HASKELL
Texto original: O mar salgado, quanto do teu sal
```

C. Cifra de Substituição Uma cifra de substituição é monoalfabética como uma cifra César, no sentido em que cada letra é sempre codificada do mesmo modo. No entanto, o método é mais complexo do que simplesmente avançar um certo número de casas. Em vez disso, constrói-se uma permutação das letras do alfabeto a partir da chave dada.
Suponhamos que conhecemos a palavra chave `PROGRAMACAO`. Começamos por remover as letras repetidas da chave, ficando com

```
PROGAMC
```

De seguida, enchemos a permutação com as letras do alfabeto que faltam.

```
PROGAMCBDEFHIJKLNQSTUVWXYZ
```

Obtemos então uma correspondência entre uma letra e a sua codificação: `A->P`, `B->R`, `C->O`, etc.

```
Alfabeto original: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Alfabeto cifrado : PROGAMCBDEFHIJKLNQSTUVWXYZ
```

Utilizamos o mesmo alfabeto cifrado para as letras minúsculas. Para codificar uma mensagem, basta ler a correspondência de cima para baixo.

```
Texto original: As armas e os baroes assinalados
Método: Substituição, chave = PROGRAMACAO
Texto cifrado : Ps pqips a ks rpqkas pssdjphpgks
```

Para decifrar uma mensagem, conhecendo a chave, basta repetir os passos de codificação, mas lemos a correspondência de baixo para cima: `P->A`, `R->B`, `O->C`, etc.

```
Texto original: K ipq sphcpgk, nupjtk gk tau sph
Método: Substituição, chave = PROGRAMACAO
Texto original: O mar salgado, quanto do teu sal
```

D. A vossa aplicação de cifra Devem produzir um ficheiro `Main.hs` que deverá conter uma função `main` de modo a poder ser executado pela linha de comandos, através de uma instrução do tipo

```
> ghc --make Main.hs
> ./Main metodo direcao chave
```

onde

- `metodo` designa um dos três métodos descritos acima. Valores possíveis: `cesar`, `vigenere`, `substitui`.
- `direcao` indica se queremos cifrar ou decifrar uma mensagem. Valores possíveis: `enc`, `dec`.
- `chave` indica a chave utilizada: um número para a cifra César, uma palavra em maíusculas para as cifras Vigenère e de substituição.

A instrução acima lê a mensagem do `stdin` e escreve a nova mensagem (cifrada ou decifrada) no `stdout`. Exemplos de execução (linhas ímpares denotam `stdin`, linhas pares denotam `stdout`):

```
> ./Main cesar enc 3
As armas e os baroes assinalados
Dv dupdv h rv edurhv dvvlqdodgrv
Que da ocidental praia Lusitana
Txh gd rflghqwdo sudld Oxvlwdqd

> ./Main vigenere dec HASKELL
V msb wlwnavy, uflutg ns epb ssv
O mar salgado, quanto do teu sal
Wlz saybmxlz dw Zscebgsv!
Sao lagrimas de Portugal!
```

Para ajudar a testar o vosso código, incluímos no Moodle da disciplina dois exemplos. Para cada exemplo, temos um ficheiro de texto original, um ficheiro texto onde aplicámos a cifra César com chave 3, um ficheiro texto onde aplicámos a cifra Vigenère com chave `HASKELL` e um ficheiro texto onde aplicámos a cifra de substituição com chave `PROGRAMACAO`. Podem usar o comando `diff` para comparar que o vosso código produz o resultado esperado.

```
> ./Main substitui enc PROGRAMACAO < texto01P.txt
Ps pqips a ks rpqkas pssdjphpgks
Nua gp kodgajtph lqpdp Husdtpjp
Lkq ipqas jujop ga pjtas jpvacpgks
Lpsspqpi pdjgp phai gp Tplqkrpjp,
Ai laqdcks a cuaqqps asmkqopgks
Ipds gk nua lqkiatdp p mkqop buipjp,
```

```
A ajtqa cajta qaiktp agdmdopqpi
Jkvk Qadjk, nua tpjtk surhdipqpi
> ./Main substitui enc PROGRAMACAO < texto01P.txt | diff texto01S.txt -
[Não produz output]
```

E. Testando a aplicação A vossa aplicação deve incluir pelo menos *sete* testes QuickCheck.

- Um teste que verifique a propriedade: usar uma cifra César com chave n, seguida de uma cifra César com chave m, é o mesmo que usar uma cifra César com chave $n + m$.
- Três testes (um para cada cifra) que verifiquem a propriedade: cifrando uma mensagem e decifrando com a mesma chave, reavemos a mensagem original.
- Outros três testes da vossa autoria. Procuramos testes interessantes, capazes de apanhar erros subtis, e não testes triviais tais como: o comprimento de uma mensagem cifrada é maior ou igual a zero.

Cada teste deverá vir acompanhado de um breve comentário que explica a propriedade a ser testada. Os testes são exercitados quando se usa a flag `-t` na linha de comandos.

```
> ./Main -t
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
```

Sugestão: se se limitarem a gerar uma **`String`** aleatória, é pouco provável que a **`String`** corresponda a uma mensagem válida (só com letras e sinais de pontuação) ou uma chave válida (só com letras). Sugerimos que definam tipos para mensagens e chaves válidas, do estilo

```
newtype MensagemValida = MensagemValida String
newtype ChaveValida = ChaveValida String
```

e que tornem os novos tipos instância da classe `Arbitrary`. Estes tipos servem apenas para gerar input válido para testes, não devendo extravazar o módulo dos testes (não devem aparecer na lista de exportação do módulo).

F. Erros na interação A vossa aplicação não se deve atrapalhar com argumentos inválidos na linha de comandos. Em vez disso deverá imprimir uma pequena explicação sobre a utilização da aplicação (*usage*). Por exemplo:

```
> ./Main olá como vais?
Utilização:
  Main -t -- Corre os testes
  Main método direçao chave -- ...
```

G. Organização em módulos A vossa aplicação deverá estar organizada em vários módulos. O modulo `Main` deverá apenas conter a parte do código que faz interação com o mundo exterior (as funções **`IO`**). Sugerimos um módulo separado para os testes e um ou mais para as cifras. A declaração de cada módulo deve listar explicitamente os tipos de dados e funções exportados (exportando apenas o que fizer sentido).

Notas

1. O vosso trabalho deverá ser constituído por um ficheiro zip de nome `t4_fcXXXXX_fcYYYYY.zip`, onde `XXXXX`, `YYYYY` são os vossos números de aluno. O ficheiro zip deverá conter no mínimo um ficheiro `Main.hs`, bem como outros módulos adicionais que julguem relevantes.

2. Os trabalhos serão avaliados semi-automaticamente. Respeitem a sintaxe das instruções para correr o executável `./Main`.

3. Cada função (ou expressão) que escreverem deverá vir sempre acompanhada de uma assinatura.

4. Lembrem-se que as boas práticas de programação Haskell apontam para a utilização de várias funções simples em lugar de uma função única mas complicada.

5. Iremos considerar os seguintes pontos para avaliar o vosso trabalho: percentagem de testes (da bateria preparada pelos docentes) passados automaticamente; legibilidade, organização e qualidade do código; qualidade das propriedades QuickCheck implementadas.

Entrega. Este é um trabalho de resolução em grupos de dois alunos. Os trabalhos devem ser entregues no Moodle até às 23:55 do dia 13 de dezembro de 2021.

Plágio. Os trabalhos de todos os grupos serão comparados por uma aplicação computacional. Relembramos aqui um excerto da sinopse: "Alunos detetados em situação de fraude ou plágio, plagiadores e plagiados, ficam reprovados à disciplina (sem prejuízo de ser acionado processo disciplinar concomitante)".

Pontos Bónus Para receber 0,2 de bónus na nota final do trabalho, siga as instruções abaixo. Pista: cada parte do texto usa uma cifra diferente.

```
Aiku oobec wl joabod jhhuf, zedxhhdn va ftmcmn tiypt xo
sqcsmblo Ziiy.pl: kunt o ywfy dn kaykti vrvcplhla qw
Fpamcvnt Efzhpifio pu wiif uiw m wyzraspbx?

Mabi xizbm li umvaiomu cai cui kqnzi lm Kmaiz. I kpidm
lmabi xizbm kwzzmaxwvlm iw biuivpw li kpidm cbqtqhili vi
xzqumqzi xizbm.

Enqp kpmqe sp gehnpreg tnp tgp ibump se ntanqbqtbipj. P
iopve senqp kpmqe sp tgp kbnqp njame p iopve tqbfbzpsp hp
kmbgebmp kpmqe.
```