Instituto Superior Técnico

Fundamentos da Programação
2016/2017

Enunciado do 2º Projecto

Data de entrega: 7 de Dezembro de 2016 às 23h59

Encriptação de Mensagens (continuação)

No 1º Projecto foi feita a implementação de um algoritmo de encriptação baseado no quadrado de Polybius. Pretende-se no 2º projecto implementar um outro algoritmo de encriptação conhecido por cifra de Playfair.

1 A cifra de Playfair

O algoritmo que vamos utilizar baseia-se na **cifra de Playfair**, também conhecida por quadrado de Playfair, cifra de Wheatstone-Playfair ou cifra de Wheatstone.[1] Esta cifra foi inventada por Charles Wheatstone em 1854 e a sua utilização foi promovida por Lorde Playfair. Foi usada por forças militares durante as 1ª e 2ª Grandes Guerras.
Para aplicar a cifra de Playfair é necessário primeiro gerar a chave de encriptação e de seguida encriptar a mensagem desejada usando a chave. Os métodos de geração de chaves de encriptação e de desencriptação são descritos de seguida.

1.1 Geração de chaves

Uma chave consiste numa tabela 5×5 preenchida com as letras consideradas. A obtenção da chave é determinada por 3 factores:

1. As *letras* consideradas (em número de 25), designadas por *letras*;
2. Uma mensagem qualquer escrita usando as 25 letras consideradas e espaços, designada por *mensagem de geração* de chave, ou simplesmente *mgc*.
3. A forma de dispôr as letras na chave, também designada *disposição*.

[1] A informação desta secção foi obtida em https://en.wikipedia.org/wiki/Playfair_cipher.

As 25 letras consideradas na cifra são as apresentadas a seguir na ordem indicada. A chave será uma tabela 5×5 refletindo determinada disposição das 25 letras consideradas.

$$letras = \{\text{A, B, C, D, E, F, G, H, I, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z}\}$$

Considere que existem apenas duas formas de dispôr letras na chave (2 disposições):

1. *Por linhas* - começando na primeira linha da tabela.
2. *Em espiral* - com início em qualquer dos cantos da tabela e com dois sentidos possíveis: no sentido dos ponteiros do relógio, e em sentido contrário.

O primeiro passo da geração de uma chave consiste em colocar a mensagem de geração, *mgc*, na tabela de acordo com a disposição escolhida (por linhas ou em espiral).
Considere a seguinte mensagem de geração de chave:

$$mgc = \text{“MENSAGEM GERACAO DE CHAVE”}$$

Aquando da colocação da mensagem na chave deverão ser colocadas na tabela as letras presentes na mensagem, na ordem em que aparecem, ignorando os espaços em branco e as letras repetidas, isto é, para todas as letras com mais que uma ocorrência só será inserida na tabela a primeira ocorrência.

Neste contexto, na **disposição por linhas** obtemos a tabela:

M	E	N	S	A
G	R	C	O	D
H	V			

Por forma a completar a chave, doravante designada `chave1`, inserem-se na tabela as letras que não ocorrem na mensagem pela ordem em que aparecem em *letras*.

M	E	N	S	A
G	R	C	O	D
H	V	B	F	I
K	L	P	Q	T
U	W	X	Y	Z

Na **disposição em espiral** no sentido dos ponteiros do relógio, seria gerada a seguinte chave, doravante designada `chave2`:

M	E	N	S	A
K	L	P	Q	G
I	Y	Z	T	R
F	X	W	U	C
B	V	H	D	O

1.2 Encriptação de mensagens

Dada uma chave, o primeiro passo da encriptação de uma mensagem é a **determinação e transformação dos *digramas*** da mensagem sem espaços (sequências de 2 letras consecutivas), da seguinte forma:

- *Se as letras do digrama forem diferentes*: o digrama mantém-se inalterado.
- *Se as letras do digrama forem iguais*: inserir um 'X' na segunda posição do digrama deslocando assim toda a mensagem uma posição para a direita.
- *Se o último digrama estiver incompleto*, por restar apenas uma letra (caso em que a mensagem tem um número impar de letras): inserir um 'X' na segunda letra do digrama.

Por exemplo, dada a mensagem "CARROS DE INIMIGO VISTA" temos:

- Mensagem sem espaços: "CARROSDEINIMIGOVISTA"
- Digramas: CA RR OS DE IN IM IG OV IS TA
- Digramas transformados: CA RX RO SD EI NI MI GO VI ST AX

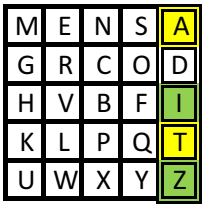
Nos digramas acima, os espaços apenas são utilizados para facilidade de leitura, não fazendo por isso parte dos digramas.

No segundo passo, procede-se à **encriptação dos digramas** usando a chave e seguindo as regras abaixo, de forma a obter a mensagem encriptada.

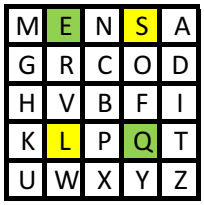
Regra 1: *Se as letras do digrama ocorrerem na **mesma linha** da chave* - substituir cada letra do digrama pela letra na mesma linha que está imediatamente à direita (coluna seguinte). Se o fim da linha for atingido, voltar ao início da linha (primeira coluna). Por exemplo, dada a `chave1`, a encriptação do digrama "VI" seria "BH". Na representação da chave que se segue as letras originais estão marcadas a verde e a sua encriptação a amarelo.

M	E	N	S	A
G	R	C	O	D
H	V	B	F	I
K	L	P	Q	T
U	W	X	Y	Z

Regra 2: *Se as letras do digrama ocorrerem na **mesma coluna** da chave* - substituir cada letra do diagrama pela letra na mesma coluna que está imediatamente abaixo (linha seguinte). Se o fim da coluna for atingido, voltar ao início da coluna (primeira linha). Por exemplo, dada a `chave1`, a encriptação do digrama "IZ" seria "TA". Na representação da chave que se segue as letras originais estão marcadas a verde e a sua encriptação a amarelo.



Regra 3: *Se as letras do digrama ocorrerem em* ***linhas e colunas diferentes*** *da chave* - considerar que as letras do digrama estão em cantos opostos de um rectângulo e substituir cada letra do digrama pela letra que está na mesma linha no canto oposto do rectângulo. Por exemplo, dada a `chave1`, a encriptação do digrama "EQ" seria "SL". Na representação da chave que se segue as letras originais estão marcadas a verde, e a sua encriptação a amarelo:



Dadas as regras atrás descritas e usando a `chave1`, a codificação da mensagem "CARROS DE INIMIGO VISTA" é "DNCWCDAOAVABAHRDBHAQNZ".

1.3 Desencriptação de mensagens

De forma a desencriptar uma mensagem encriptada de acordo com as regras apresentadas anteriormente, deverão ser aplicadas as *regras inversas às regras 1 e 2*, e a *regra 3*. Note que a mensagem desencriptada não corresponde à mensagem original, mas sim aos digramas transformados desta mensagem. Por exemplo, a desencriptação da mensagem encriptada "DNCWCDAOAVABAHRDBHAQNZ" é "CARXROSDEINIMIGOVISTAX".

2 Tipos Abstractos de Dados (TAD)

Deverá definir os seguintes tipos abstractos de dados. Para cada tipo, apenas os construtores devem verificar a validade dos argumentos. A implementação das funções básicas descritas é obrigatória. Para além destas, poderá definir outras que julgue conveniente.

2.1 Tipo posicao (2.0 valores)

Uma posição é um tuplo de 2 inteiros não negativos correspondentes a uma linha e uma coluna.

Operações básicas:

1. *Construtor*:
   - $faz\_pos : \mathbb{N}^0 \times \mathbb{N}^0 \mapsto posicao$
     $faz\_pos(l, c)$ recebe dois argumentos do tipo inteiro, uma linha l e uma coluna c, e devolve um elemento do tipo *posicao*.
2. *Seletores*:
   - $linha\_pos : posicao \mapsto \mathbb{N}^0$
     $linha\_pos(p)$ recebe um argumento do tipo *posicao* e devolve a linha respectiva.
   - $coluna\_pos : posicao \mapsto \mathbb{N}^0$
     $coluna\_pos(p)$ recebe um argumento do tipo *posicao* e devolve a coluna respectiva.
3. *Reconhecedores*:
   - $e\_pos : universal \mapsto Boolean$
     $e\_pos(arg)$ devolve $True$ se o argumento arg for do tipo *posicao* e $False$ caso contrário.
4. *Testes*:
   - $pos\_iguais : posicao \times posicao \mapsto Boolean$
     $pos\_iguais(p_1, p_2)$ recebe dois argumentos do tipo *posicao*, p_1 e p_2, e devolve $True$ caso os argumentos correspondem à mesma posição da chave e $False$ caso contrário.

2.2 Tipo chave (4.0 Valores)

Uma chave consiste numa tabela 5×5. Cada elemento da tabela deve ser uma letra maiúscula do conjunto
$L = \{A, B, C, D, E, F, G, H, I, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z\}$.

Operações básicas:

1. *Construtores*:

   - $gera\_chave\_linhas : L \times str \mapsto chave$
     $gera\_chave\_linhas(l, mgc)$ recebe dois argumentos, l e mgc, correspondentes a um tuplo de 25 letras e à cadeira de caracteres mgc e devolve a chave gerada usando a disposição por linhas (descrito na pág. 2).
   - $gera\_chave\_espiral : L \times str \times \{$ 'r', 'c'$\} \times posicao \mapsto chave$
     $gera\_chave\_espiral(l, mgc, s, pos)$ recebe 4 argumentos, em que l corresponde a um tuplo de 25 letras e mgc à mensagem de geração de chave, e devolve a chave gerada usando a disposição em espiral no sentido indicado pelo parâmetro s ('r' para o sentido dos ponteiros do relógio e 'c' para o sentido contrário), começando na posição indicada pelo parâmetro pos, tal como descrito na pág. 2.

2. *Seletor*:

   - $ref\_chave : chave \times posicao \mapsto L$
     $ref\_chave(c, p)$ recebe como argumentos a chave c e a posição p e devolve a letra que está em c na posição p.

3. *Modificador*:

   - $muda\_chave : chave \times posicao \times L \mapsto chave$
     $muda\_chave(c, p, l)$ recebe como argumentos a chave c, a posição p e a letra l e devolve a chave c com a letra l na posição p.

4. *Reconhecedores*:

   - $e\_chave(arg)$
     devolve $True$ se o argumento arg for do tipo $chave$ e $Falso$ caso contrário.

5. *Transformadores*:

   - $string\_chave : chave \mapsto str$
     $string\_chave(c)$ devolve uma cadeia de caracteres que uma vez impressa apresenta as letras de c dispostas numa tabela 5×5 (ver exemplos na secção seguinte).

3 Exemplos

Nesta secção apresentam-se alguns exemplos de utilização das operações básicas.

```
>>> letras = ('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M',
        'N',  'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z')
>>> letras_erradas = ('A', 'A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L',
        'M', 'N',  'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z')
>>> mgc = "MENSAGEM GERACAO DE CHAVE"
>>> chave1 = gera_chave_linhas(letras, mgc)
>>> chave2 = gera_chave_espiral(letras, mgc, 'r', faz_pos(0,0))
>>> gera_chave_linhas(letras_erradas, mgc)
                  ...
builtins.ValueError: gera_chave_linhas: argumentos errados
>>> ref_chave(chave1, faz_pos(1,3))
'O'
>>> e_chave(chave1)
True
>>> string_chave(chave1)
'M E N S A \nG R C O D \nH V B F I \nK L P Q T \nU W X Y Z \n'
>>> string_chave(chave2)
'M E N S A \nK L P Q G \nI Y Z T R \nF X W U C \nB V H D O \n'
>>> print(string_chave(chave1))
M E N S A
G R C O D
H V B F I
K L P Q T
U W X Y Z

>>> print(string_chave(chave2))
M E N S A
K L P Q G
I Y Z T R
F X W U C
B V H D O

>>> print(string_chave(muda_chave(chave1, faz_pos(0,0), '?')))
? E N S A
G R C O D
H V B F I
K L P Q T
U W X Y Z

>>> e_chave(chave1)
False
```

4 Funções a desenvolver (10.0 Valores)

Nesta secção são descritas as funções que deverá desenvolver de forma a tornar possível a encriptação e desencriptação de mensagens de acordo com os métodos descritos nas Secções 1.2 e 1.3. Nenhuma das funções necessita verificar a validade dos argumentos.

(2.0 Valores) Escreva a função `digramas` que recebe como argumento uma cadeia de caracteres correspondente a uma mensagem, `mens`, e devolve a cadeia de caracteres correspondente aos digramas transformados de `mens` sem espaços (ver pág. 3).
Por exemplo:

```
>>> digramas('CARROS DE INIMIGO VISTA')
'CARXROSDEINIMIGOVISTAX'
```

(2.0 Valores) Escreva a função `figura` que recebe dois argumentos, `digrm`, uma cadeia de caracteres de comprimento 2, e `chave`, uma chave, e devolve um tuplo de 3 elementos da forma `(fig, pos1, pos2)` em que:

- `fig` é a figura determinada pelas letras de `digrm`, 'l', 'c' ou 'r' (linha, coluna ou rectângulo).
- `pos1` e `pos2` são as posições ocupadas na chave pela primeira e segunda letras de `digrm`, respectivamente.

Por exemplo:

```
>>> figura('VI', chave1)
('l', (2, 1), (2, 4))
>>> figura('IZ', chave1)
('c', (2, 4), (4, 4))
>>> figura('EQ', chave1)
('r', (0, 1), (3, 3))
```

(1.0 Valores) Escreva a função `codifica_l` (de "codifica linha") que recebe três argumentos, `pos1`, `pos2`, consistindo nas posições das letras de um digrama na mesma linha de uma chave, e o inteiro `inc`, que poderá ser 1 ou -1. Se `inc` for 1 pretende-se fazer uma encriptação; se `inc` for -1 pretende-se fazer uma desencriptação.
A função devolve um tuplo de 2 posições `(pos1_cod, pos2_cod)` que correspondem às posições das letras do digrama encriptado/desencriptado.
Por exemplo:

```
>>> codifica_l((2,1),(2,4), 1)
((2, 2), (2, 0))
>>> codifica_l((2,1),(2,4), -1)
((2, 0), (2, 3))
```

(1.0 Valores) Escreva a função `codifica_c` (de "codifica coluna") que recebe três argumentos, `pos1`, `pos2`, consistindo nas posições das letras de um digrama na mesma coluna de uma chave, e o inteiro `inc`, que poderá ser 1 ou -1. Se `inc` for 1 pretende-se fazer uma encriptação; se `inc` for -1 pretende-se fazer uma desencriptação. A função devolve um tuplo de 2 posições `(pos1_cod, pos2_cod)` que correspondem às posições das letras do digrama encriptado/desencriptado.
Por exemplo:

```
>>> codifica_c((2,4),(4,4), 1))
((3, 4), (0, 4))
>>> codifica_c(((2,4),(4,4), -1)
((1, 4), (3, 4))
```

(1.0 Valores) Escreva a função `codifica_r` (de "codifica rectângulo") que recebe dois argumentos, `pos1`, `pos2`, consistindo nas posições das letras de um digrama numa chave. Estas posições encontra-se em linhas e colunas diferentes. A função devolve um tuplo de 2 posições `(pos1_cod, pos2_cod)` que correspondem às posições das letras do digrama encriptado/desencriptado.
Por exemplo:

```
>>> codifica_r((0,1), (3,3))
((0, 3), (3, 1))
```

(2.0 Valores) Escreva a função `codifica_digrama` que recebe três argumentos, `digrm`, um digrama, `chave`, uma chave, e o inteiro `inc`, que poderá ser 1 ou -1. Se `inc` for 1 pretende-se fazer uma encriptação; se `inc` for -1 pretende-se fazer uma desencriptação. A função devolve o digrama correspondente à encriptação/desencriptacão de `digrm` usando a `chave`.
Por exemplo:

```
>>> codifica_digrama('VI', chave1, 1)
'BH'
>>> codifica_digrama('EQ', chave1, -1)
'SL'
```

(1.0 Valores) Escreva a função `codifica` que recebe três argumentos, `mens`, uma mensagem, `chave`, uma chave, e o inteiro `inc`, que poderá ser 1 ou -1. Se `inc` for 1 pretende-se fazer uma encriptação; se `inc` for -1 pretende-se fazer uma desencriptação. A função devolve a mensagem correspondente à encriptação/desencriptacão de `mens` usando a `chave`.
Por exemplo:

```
>>> codifica('CARROS DE INIMIGO VISTA', chave1, 1)
'DNCWCDAOAVABAHRDBHAQNZ'
>>> codifica('DNCWCDAOAVABAHRDBHAQNZ', chave1, -1)
'CARXROSDEINIMIGOVISTAX'
```

5 Avaliação

O projecto é individual e a avaliação do projecto terá duas componentes:

1. **Avaliação automática (16 valores)**;
2. **Avaliação manual (4 valores)**.

Na **avaliação automática**, que avaliará a **execução do programa**, será usado o sistema **Mooshak** que testará o programa usando um conjunto de **testes privados** seguindo as cotações indicadas no enunciado para cada função a desenvolver. Não será disponibilizado qualquer tipo de informação sobre os casos de teste privados utilizados pelo sistema na avaliação do projecto. Estes serão disponibilizados na página da disciplina após a entrega.

Uma semana antes do prazo de entrega serão disponibilizados um conjunto de **testes públicos** que deve usar para testar o funcionamento do programa antes de o submeter para avaliação no Mooshak. No caso dos testes públicos será disponibilizado o input do programa e o respectivo output para que possa detectar eventuais erros semânticos.

Na **avaliação manual**, que avaliará a **qualidade do código** desenvolvido, serão usados os seguintes critérios e cotações:

- Abtracção procedimental, tamanho das funções e reutilização de código **(1 valor)**;
- Abtracção de dados (respeito das barreiras de abstracção) **(1 valor)**;
- Clareza dos nomes das variáveis e das funções **(1 valor)**;
- Qualidade (e não quantidade) dos comentários **(1 valor)**.

6 Condições de Realização e Prazos

A entrega do 1º projecto será efectuada exclusivamente por via electrónica. Deverá submeter o seu projecto **através do sistema Mooshak** até às **23:59** do dia **7 de Dezembro de 2016**. Projectos em atraso não serão aceites seja qual for o pretexto.

Deverá submeter um **único ficheiro com extensão .py** contendo todo o código do seu projecto (implementação das funções pedidas). O ficheiro de código deve conter em **comentário na primeira linha o número e o nome do aluno**.

No ficheiro de código não podem ser utilizados caracteres acentuados ou qualquer caractere que não pertença à tabela ASCII. Isto inclui comentários e cadeias de caracteres. Programas que não cumpram este requisito serão penalizados em três valores.

Uma semana antes do prazo de entrega *serão publicadas na página da disciplina as instruções necessárias para a submissão do código no Mooshak. Apenas a partir dessa altura será possível a submissão por via electrónica. Nessa altura serão também fornecidas as credenciais de acesso individuais. Até ao prazo de entrega poderá efectuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a última entrega efectuada. Deverá portanto verificar cuidadosamente que a versão submetida no Mooshak corresponde à versão do projecto que pretende que seja avaliada. Não serão abertas excepções.*

Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados no enunciado e/ou nos testes públicos, além de um conjunto de testes adicionais. O facto de um projecto completar com sucesso os exemplos fornecidos no enunciado não implica, pois, que esse projecto esteja totalmente correcto, pois o conjunto de exemplos fornecido não é exaustivo. **É da responsabilidade do programador garantir que o código produzido está correcto**.

Caso o corpo docente considere necessário poderá existir uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (decidida caso a caso).

Projectos iguais, ou muito semelhantes, serão penalizados com a reprovação à disciplina. O corpo docente da disciplina será o único juiz o que se considera ou não copiar num projecto. **Será usado um sistema automático de detecção de cópias**.

7 Aspectos a evitar

Os seguintes aspetos correspondem a sugestões para evitar maus hábitos de trabalho (e, consequentemente, más notas no projeto):

1. Não pense que o projeto se pode fazer nos últimos dias. Se apenas iniciar o seu trabalho neste período irá ver a Lei de Murphy em funcionamento (todos os problemas são mais difíceis do que parecem; tudo demora mais tempo do que nós pensamos; e se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis).

2. Não duplique código. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos.

3. Não se esqueça que as funções excessivamente grandes são penalizadas no que respeita ao estilo de programação.

4. A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente errada.

5. Quando o programa gerar um erro, preocupe-se em descobrir qual a causa do erro. As "marteladas" no código têm o efeito de distorcer cada vez mais o código.

6. Utiliza os teste públicos para testar o programa e corrigir eventuais erros. Se o programa errar um teste público vai de certeza errar pelo menos um teste privado usando pelo Mooshak.

7. Não espere pelo final do prazo, altura em que o sistema estará sobrecarregado, para fazer a submissão ao Mooshak. Nessa altura pode não conseguir submeter o projecto e a responsabilidade será sua.

8. Não espere até ter implementado todo o projecto para fazer a primeira submissão ao Mooshak. Submeta versões incrementais. Se o programa já passa alguns dos testes públicos também passará alguns dos testes privados.