

Esteganografia

Neste exercício-programa, você ganhará mais familiaridade com manipulação de matrizes e também trabalhará com imagens digitais. Este EP trata de uma técnica usada para transmissão secreta de informação, chamada **esteganografia**.¹ Nesta técnica, uma mensagem é escondida em uma imagem.

1 Esteganografia

Considere uma imagem, digamos D , em preto e branco, formada por diversos “pixels” em níveis de cinza, sendo que o tom de cada pixel varia de 0 (zero = preto) a 255 (branco). Se alteramos alguns pixels “espalhados” de D “levemente”, a mudança na imagem é imperceptível a olho nu. Assim, dado um texto T , podemos “espalhar” os bits que representam T ² na imagem D e assim “esconder” T dentro de D . Seja D^0 essa pequena modificação de D em que T está escondida. Se enviamos D^0 a alguém, ela poderá recuperar o texto T a partir de D^0 e D (naturalmente, desde que ela saiba os detalhes do algoritmo que usamos). Assim, conseguimos transmitir T secretamente, sem ninguém perceber.

Para fazer essa “pequena alteração” dos pixels de D , especificamos um parâmetro inteiro b (tipicamente, $b = 1$ ou 2) e somente alteramos os b bits menos significativos dos pixels escolhidos para alteração.³ Descrevemos a seguir a técnica mais formalmente.

Seja D uma matriz $m \times n$ de inteiros, representando uma imagem. Cada elemento da matriz é um inteiro (chamado de *pixel*) que codifica um tom de cinza (que varia de 0 a 255, com 0 representando preto e 255 branco). Cada pixel é classificado como

¹Veja <https://pt.wikipedia.org/wiki/Esteganografia>.

²Lembre que tudo que pode ser armazenado na memória do computador é armazenado como uma sequência de bits.

³Veja https://en.wikipedia.org/wiki/Binary_number#Binary_counting para a representação de números em base binária.

- **pixel de ocultação**, se ele é usado para ocultar o texto;
- **pixel intocável**, se ele não é modificado no processo de esteganografia.

Seja T um texto com $k > 0$ caracteres, onde cada caractere é representado por um byte (8 bits)⁴. Queremos obter uma matriz D^0 , de dimensões $m \times n$, que represente o desenho com o texto T ocultado por esteganografia. Sejam b e d inteiros positivos fixos (esses inteiros farão parte da especificação do algoritmo a seguir). Um pixel de ocultação será reservado para codificar os valores de b e d . Note que, portanto, temos $mn - 1$ pixels disponíveis para ocultar T (um será gasto para codificar b e d e temos um total de mn pixels). O parâmetro b designa o *número de bits menos significativos* a serem usados em cada pixel de ocultação. Como T tem k caracteres e cada caractere gasta 8 bits, é necessário que

$$(mn - 1)b \geq 8k. \quad (1)$$

Ademais, será conveniente supor que

$$b \text{ é um divisor positivo de } 8. \quad (2)$$

Agora, é possível que $(mn - 1)b$ seja muito maior que $8k$, de forma que podemos usar muito menos que $mn - 1$ pixels de ocultação para esconder T . Nesse caso, podemos “espalhar” os pixels de ocultação pela imagem. Procedemos da seguinte forma. Definimos d como sendo o *espaçamento entre linhas e colunas* dos pixels de ocultação; assim, os pixels de ocultação ficam sendo aqueles com as seguintes coordenadas:⁵

$$\begin{array}{ccccccc} (d-1, d-1), & (d-1, 2d-1), & \dots, & (d-1, b_d^m cd - 1), \\ (2d-1, d-1), & (2d-1, 2d-1), & \dots, & (2d-1, b_d^m cd - 1), \\ \vdots & \vdots & \ddots & \vdots \\ (b_d^m cd - 1, d-1), & (b_d^m cd - 1, 2d-1), & \dots, & (b_d^m cd - 1, b_d^m cd - 1). \end{array} \quad (3)$$

Observação. As coordenadas (i, j) dos pixels são tais que i varia de 0 a $m - 1$ e j varia de 0 a $n - 1$ (exatamente como em uma matriz $m \times n$).

Note que, no conjunto de coordenadas em (3), temos $b_d^m cd$ pixels de ocultação no desenho, cada um deles ocultando b bits de informação em seus b bits menos significativos, o que resulta em um total de $b_d^m cb$ bits disponíveis para ocultação.

Usaremos os b bits menos significativos do pixel em $(d - 1, d - 1)$ para codificar o valor de b ⁶

⁴A codificação mais comumente encontrada para os caracteres é a dada pela tabela ASCII, que de fato define apenas códigos de 0 a 127 (7 bits). Existem várias extensões, como a latin1 (ISO-8859-1), que codificam letras acentuadas nos números de 128 a 255.

⁵Relembramos que, dado x um real qualquer, bxc denota o **maior inteiro menor ou igual a** x . Assim, $b_d^m c$ denota o quociente inteiro da divisão de m por d . Em \mathbb{C} , isto coincide com a divisão de duas variáveis inteiras não negativas.

⁶Note que b pode ser representado com b bits.

e precisamos de $8k$ bits para T . Segue que d deve ser tal que

$$\mathbf{j} \frac{m}{d} \mathbf{k} \mathbf{j} \frac{n}{d} \mathbf{k} b \geq 8k + b. \quad (4)$$

Também é necessário que

$$1 \leq d \leq \min(m, n). \quad (5)$$

Quanto aos bits de T a serem ocultados, são considerados primeiramente os b bits menos significativos do caractere $T[0]$, depois os b bits seguintes, até que finalmente tenhamos considerado os b bits mais significativos de $T[k-1]$. Usando a suposição que b é um divisor de 8 (lembre-se de (2)), a sequência de números de b bits a serem ocultados no desenho é⁷

$$b, \quad \begin{array}{c} \mathbf{j} \frac{T[0]}{2^0} \mathbf{k} \\ \mathbf{j} \frac{T[1]}{2^0} \mathbf{k} \\ \vdots \\ \mathbf{j} \frac{T[k-1]}{2^0} \mathbf{k} \end{array} \bmod 2^b, \quad \begin{array}{c} \mathbf{j} \frac{T[0]}{2^b} \mathbf{k} \\ \mathbf{j} \frac{T[1]}{2^b} \mathbf{k} \\ \vdots \\ \mathbf{j} \frac{T[k-1]}{2^b} \mathbf{k} \end{array} \bmod 2^b, \quad \dots \quad \begin{array}{c} \mathbf{j} \frac{T[0]}{2^{8-b}} \mathbf{k} \\ \mathbf{j} \frac{T[1]}{2^{8-b}} \mathbf{k} \\ \vdots \\ \mathbf{j} \frac{T[k-1]}{2^{8-b}} \mathbf{k} \end{array} \bmod 2^b.$$

Observe que se x é o tamanho desta sequência, podemos deixar inalterados os $bm/dcbn/dc - x$ últimos pixels de ocultação de D .

Exemplo de ocultação em um pixel

Damos aqui um exemplo de ocultação de b bits em um pixel. Sejam $x, z \in \{0, 1, \dots, 255\}$ e $y \in \{0, 1, \dots, 2^b - 1\}$ tais que

- x é um tom de cinza de um pixel na imagem original D ,
- y é um inteiro de b bits que queremos ocultar naquele pixel e
- z é um tom de cinza do mesmo pixel na imagem D^0 , que foi alterado para “conter” y .

Podemos definir z da seguinte forma:

$$z = \mathbf{j} \frac{x}{2^b} \mathbf{k} 2^b + (x + y) \bmod 2^b. \quad (6)$$

Um pouco de aritmética modular implica que podemos obter y a partir de x e z ; de fato, temos

$$y = (z - x) \bmod 2^b. \quad (7)$$

Em C, para aplicar a fórmula (7), faça

$$y = (z - x + 256) \% B;$$

⁷ $A \bmod B$ é o resto da divisão inteira de A por B .

com $B = 2^b$. Na Tabela 1 temos alguns valores possíveis de x , y e z .

x	y	z
1101 0110	0100	1101 1010
0101 1010	0101	0101 1111
0101 1011	0101	0101 0000
1011 0011	0000	1011 0011

Tabela 1: Exemplos de ocultação de y num tom de cinza x resultando no tom z ($b = 4$; números em notação binária).

2 Formato PGM

Neste EP utilizaremos o formato PGM para armazenar imagens em arquivos. Segundo este formato, o arquivo deve conter um cabeçalho e a matriz correspondente à imagem. Veja um exemplo a seguir.

```
P2
5 4
16
9 4 5 0 8
10 3 2 1 7
9 1 6 3 15
1 16 9 12 7
```

A primeira linha do arquivo contém uma palavra-chave “P2”, que é obrigatória. A segunda linha contém dois números que correspondem ao número de colunas e linhas da matriz, respectivamente. A terceira linha contém um número que é **maior ou igual a todos os números** da imagem (MaxVal). Para fins deste EP, MaxVal é no máximo 255. Os demais números do arquivo correspondem aos tons de cinza da imagem armazenados em forma de uma matriz de inteiros. Cada tom de cinza é um número entre 0 e MaxVal, com 0 indicando “preto” e Maxval indicando “branco”.

O formato PGM também permite colocar comentários. Caracteres após o caractere ‘#’ até o próximo fim de linha (caractere ‘\n’) são comentários e são ignorados. Um exemplo de imagem com comentários:

```
P2
# imagem: exemplo.pgm
5 4
16
9 4 5 0 8
10 3 2 1 7
9 1 6 3 15
1 16 9 12 7
```

Neste EP, por simplicidade, *você pode supor que os arquivos que serão manipulados não contêm comentários*. De fato, vamos fornecer um programa que “filtra” (joga fora) tais comentários. Assim, se você quiser trabalhar com imagens com comentários, você pode primeiro “filtrar” esses comentários e criar um novo arquivo, sem comentários, que pode então ser usado em seu EP.

3 Detalhamento do EP

Neste EP você deve implementar as funções abaixo, como descrito em suas definições. Você pode, se desejar, implementar outras funções para resolver o EP. Tenha em mente que seu programa deverá operar em dois modos, um dos quais é chamado **verborrágico**, no qual se imprime mais coisas na saída.

Adote

```
#define MAX      400
#define MAX2    160000
#define FNMAX    200
```

1) Escreva em C uma função LeDesenho de protótipo

```
int LeDesenho( char nomearq[FNMAX], int M[MAX][MAX],
               int *pm, int *pn, int *pmax );
```

A string nomearq guarda o nome de um arquivo em formato PGM. Devem ser devolvidos em *pm, *pn, e *pmax os valores do número de linhas, de colunas e o valor máximo que codifica um tom de cinza do arquivo de nome nomearq, respectivamente. O desenho que está contido no arquivo deve ser devolvido na matriz M. A função deve devolver 0 se não houver nenhum erro, e deve devolver 1 caso algum erro tenha sido encontrado (ou por conta da manipulação do arquivo, ou por conta de o valor de MAX ser insuficiente).

2) Escreva em C uma função LeTexto de protótipo

```
int LeTexto( char nomearq[FNMAX], char T[MAX2], int *pk );
```

A string nomearq contém o nome de um arquivo. O texto presente nele deverá ser devolvido em T. Deve-se devolver em *pk o número de caracteres lidos e armazenados em T. A função deve devolver 0 se não houver nenhum erro, e deve devolver 1 caso algum erro tenha sido encontrado (ou por conta da manipulação do arquivo, ou por conta de o valor de MAX2 ser insuficiente).

3) Escreva em C uma função BeDe de protótipo

```
int BeDe( int k, int m, int n, int *pb, int *pd );
```

A função deve calcular os parâmetros b e d do processo de ocultação por esteganografia de um texto de k caracteres através de uma imagem de dimensões m por n . O parâmetro b deve ser devolvido em $*pb$. Dentre as diversas opções possíveis que satisfaçam as equações (1) e (2), deve-se escolher aquela que minimiza b (intuitivamente, dessa forma, a imagem resultante é mais parecida com a imagem original). O parâmetro d deve ser devolvido em $*pd$. Para tornar a alteração o mais imperceptível possível, busca-se espalhar ao máximo os pixels de ocultação. Assim, uma vez escolhido b mínimo, deve-se escolher d máximo dentre aqueles que satisfazem às equações (4) e (5). A função BeDe deve devolver 0 se for possível encontrar b e d válidos e deve devolver 1 caso contrário.

Na Tabela 2 temos exemplos de alguns valores obtidos para b e d em função de k , m , e n . No último exemplo, não há valores para b e d que satisfaçam as restrições necessárias.

m	n	k	b	d
7	7	1	1	2
7	14	1	1	2
7	15	1	1	3
7	7	2	1	1
7	7	6	1	1
7	7	7	2	1
7	7	12	2	1
7	7	13	4	1
7	7	24	4	1
7	7	25	8	1
7	7	48	8	1
7	7	49	—	—

Tabela 2: Exemplos de cálculos de b e d pela função BeDe

4) Escreva em C uma função ProximosBBits de protótipo

```
int ProximosBBits( char T[MAX2], int b, int *pik, int *pib );
```

A função recebe o texto T e devolve o inteiro formado pelos próximos b bits ($b = b$) do texto T . Estes próximos b bits são obtidos da seguinte forma: descartam-se os $*pib$ bits menos significativos de $T[*pik]$ e tomam-se os próximos⁸ b bits menos significativos de $T[*pik]$. Os valores de $*pik$ e $*pib$ deverão ser devidamente atualizados para apontar para os próximos b bits a serem extraídos do texto T . A função deve devolver o número binário com b bits extraído do texto T .

⁸Se b não fosse um divisor de 8, poderia ocorrer que em um certo momento teriam restado em $T[*pik]$ um número r de bits menor que b . Neste caso, estes r bits seriam os menos significativos e os demais $b - r$ bits seriam os $b - r$ bits menos significativos de $T[*pik + 1]$.

Na Tabela 3 temos exemplos de alguns valores devolvidos pela função. Em todos os casos, o caractere representado por T[0] é a letra 'g', ou seja, 103 = 0110 0111 de acordo com a tabela ASCII. O valor devolvido pela função aparece na coluna proximos.

Entrada			Saída		
b	*pik	*pib	*pik	*pib	proximos
1	0	0	0	1	1
1	0	2	0	3	1
1	0	3	0	4	0
1	0	7	1	0	0
2	0	0	0	2	3
2	0	2	0	4	1
2	0	4	0	6	2
2	0	6	1	0	1
4	0	0	0	4	7
4	0	4	1	0	6
8	0	0	1	0	103

Tabela 3: Exemplos de valores calculados pela função ProximosBBits

5) Escreva em C uma função Codifica de protótipo

```
void Codifica( int D[MAX][MAX], int m, int n, char T[MAX2], int k,
              int DI[MAX][MAX], int b, int d, int modo );
```

A função recebe uma imagem na matriz D, de dimensões m por n, recebe um texto T de k caracteres, recebe em b e em d os parâmetros b e d acima descritos e recebe em modo o modo de operação corrente (se verborrágico ou não). Esta função deve determinar a imagem com esteganografia que codifica T e deve armazená-lo na matriz DI, de dimensões m por n. **Se o modo de operação corrente for verborrágico, deve imprimir alguns dados (descritos mais adiante) durante o processo de codificação.**

6) Escreva em C uma função Maximo de protótipo

```
int Maximo( int D[MAX][MAX], int m, int n );
```

A função recebe uma matriz D, de dimensões m por n, e devolve o máximo da matriz.

7) Escreva em C uma função EscreveDesenho de protótipo

```
int EscreveDesenho( char nomearq[FNMAX], int M[MAX][MAX],
                   int m, int n, int max );
```

Ela recebe em nomearq uma string com o nome de um arquivo, abre-o para escrita e coloca no arquivo nomearq o desenho da matriz M com m linhas e n colunas, usando o formato PGM. O

valor max corresponde ao máximo valor de um tom de cinza, e deve ser este o valor a ser escrito no arquivo PGM na linha correspondente. A função deve devolver 0 se não houver nenhum erro e 1 caso algum erro tenha sido encontrado por conta da manipulação do arquivo.

8) Escreva em C uma função DeBeDe de protótipo

```
void DeBeDe( int D[MAX][MAX], int DI[MAX][MAX],  
            int m, int n, int *pb, int *pd );
```

A função recebe dois desenhos representados pelas matrizes D e DI, ambas de tamanho m por n. Sabe-se que DI foi obtida de D através de uma ocultação de texto por esteganografia. A função deve detectar o parâmetro *d* que foi usado na ocultação e devolvê-lo em *pd. Da mesma forma, a função deve detectar o parâmetro *b* que foi usado no processo e deve devolvê-lo em *pb. *Sugestão*: calcule o menor *i* para o qual a diferença $DI[i][i] - D[i][i]$ não seja nula.

9) Escreva em C uma função DeCodifica de protótipo

```
int DeCodifica( int D[MAX][MAX], int DI[MAX][MAX], int m, int n,  
               int b, int d, char T[MAX2], int modo );
```

DeCodifica recebe nas matrizes D e DI, de dimensões m por n, duas imagens. A imagem em DI foi obtida a partir de D por esteganografia, com parâmetros b e d. Esta função deve devolver em T o texto que foi ocultado neste processo e deve devolver *k*, o número de caracteres em T. ***Se o modo de operação recebido em modo for verborrágico, a função deve imprimir alguns dados (descritos mais adiante).***

10) Escreva em C uma função EscreveTexto de protótipo

```
int EscreveTexto( char nomearq[FNMAX], char T[MAX2], int k );
```

A string nomearq contém o nome de um arquivo, que deverá receber uma cópia do texto T de k caracteres. A função deve devolver 0 se não houver nenhum erro, e deve devolver 1 caso algum erro tenha sido encontrado por conta da manipulação do arquivo.

11) Escreva em C um programa que peça uma das operações abaixo descritas e as execute. Isto deve ser feito repetidamente até que a operação o seja solicitada.

A **operação** 0 termina o programa.

A **operação** 1 pede, na ordem:

1. o nome de um arquivo que contém um desenho *D*;
2. o nome de um arquivo que contém um texto *T*;

3. o nome de um arquivo que deverá conter o desenho D^0 com texto ocultado por esteganografia.

Este desenho D^0 é o desenho obtido a partir de D através da ocultação do texto T por esteganografia. No arquivo em formato PGM que contém D^0 , o valor do máximo valor de um tom de cinza (MaxVal; veja a Seção 2) deve ser o máximo entre (a) o valor de MaxVal no arquivo PGM do desenho lido D e (b) o maior inteiro na matriz D^0 .

A **operação 2** pede, na ordem:

1. o nome de um arquivo que contém um desenho D ;
2. o nome de um arquivo que contém o desenho D^0 obtido por esteganografia;
3. o nome de um arquivo que deverá conter o texto T obtido da decodificação de D e D^0 .

A **operação 3** pede o nome de um arquivo que contém um texto e imprime o texto na tela.

A **operação 4** deve trocar o modo de operação para verborrágico. Uma nova operação 4 faz voltar ao modo de operação normal.

Para facilitar a correção, **algumas impressões na tela devem ser feitas:**

1. Toda vez que um desenho D (ou D^0) for lido com sucesso, os valores de m e n devem ser impressos, nesta ordem;
2. Toda vez que os parâmetros b e d forem calculados com sucesso, eles deverão ser impressos, nesta ordem;
3. Toda vez que um texto T for lido com sucesso de um arquivo ou decodificado, o número de caracteres k em T deve ser impresso.

Se o modo de operação for verborrágico, e somente neste caso, **algumas impressões na tela devem ser feitas em uma única linha, e nesta ordem, toda vez que alguns bits de texto forem codificados ou decodificados em algum pixel de ocultação:**

1. A posição escolhida que foi alterada;
2. Cada inteiro que codifica os b bits do texto que foram ocultados;
3. O tom de cinza do desenho original;
4. O tom de cinza do desenho com alteração.

As impressões dos tons de cinza devem ser feitas com formato `%02x`, para impressão de inteiros em formato hexadecimal.⁹

O programa deve começar com o modo de operação verborrágico desativado.

⁹Em notação hexadecimal, os números são denotados em base 16. Tipicamente, os dígitos hexadecimais são os dígitos de 0 a 9 mais as letras de a a f

4 Leitura e impressão de arquivos

Aqui descrevemos uma receita para fazer leitura e impressão em arquivos. Será sempre necessário fazer

```
#include <stdlib.h>
```

no topo do arquivo, junto ao

```
#include <stdio.h>
```

Não importa a ordem.

Para ler um nome de arquivo declarado com `char nomearq[FNMAX]`, utilize o comando `scanf("%s", nomearq);`

Se desejar imprimir o nome do arquivo, utilize algo como

```
printf("bla bla %s bla", nomearq);
```

Para abrir um arquivo com nome guardado em `char nomearq[FNMAX]`, primeiro declare uma variável no topo da função:

```
FILE *fp;
```

Depois, abra o arquivo com

```
if (!(fp = fopen(nomearq, "r"))) {  
    printf("Erro ao abrir o arquivo %s\n", nomearq);  
    /* mais comandos para lidar com o erro na abertura do arquivo */  
}
```

Se o programa entrar no `if`, houve um erro ao abrir o arquivo, e o programa deve seguir de forma apropriada (reportando o erro e desistindo, se for o caso). Se não entrar no `if`, o arquivo estará “aberto para leitura em `fp`”. Neste caso, você pode ler do arquivo, como se o usuário estivesse digitando cada caractere do arquivo na entrada padrão. Para fazer uma leitura de um inteiro de `fp` numa variável `int x`, faça

```
fscanf(fp, "%d", &x);
```

Naturalmente, outros formatos de leitura são aceitos. Basta sempre trocar “`scanf()`” por “`fscanf(fp,)`”.

A mesma regra segue para impressão/escrita num arquivo. Ou seja, toda ocorrência de “`printf()`” deve ser trocada por “`fprintf(fp,)`”. Além disso, na abertura do arquivo com a chamada à função `fopen`, no lugar de passar o segundo parâmetro “`r`”, deve ser passado o parâmetro “`w`”. Isso fará com que o arquivo seja “aberto para escrita em `fp`”.

Por fim, ao acabar de ler ou escrever em um arquivo, o programa deve fazer uma chamada `fclose(fp);`

para “fechar” o arquivo.