MAC–IME–USP

Carlos Eduardo Ferreira
Sala 108C Tel.: 3091 6079
e-mail cef@ime.usp.br
monitor Stefano Tommasini
e-mail stefanolord@gmail.com
monitor Jesus Alejandro Peña Mesias
e-mail jpenam@ime.usp.br

MAC 110 – Introdução à Computação – BCC

Primeiro semestre de 2013

Ocultação por marca d'água – **Entrega: 17 de junho de 2013**

O objetivo deste exercício-programa é manipulação de matrizes. Trata-se da implementação de uma técnica bastante usada em criptografia, chamada de **ocultação por marca d'água**. Nesta técnica uma mensagem é enviada de forma codificada utilizando um documento aparentemente banal para fazer o envio.

1 Ocultação por marca d'água

A ideia da técnica é ocultar alguma mensagem em um arquivo aparentemente banal, como uma imagem, por exemplo. Pela sua semelhança às marcas d'água, muitas vezes existentes em papel moeda e outros tipos de documentos, acabou levando esse nome.

Considere uma imagem em preto e branco, formada por diversos "pixels" em níveis de cinza, sendo que o tom de cada pixel varia do 0 (zero = branco) até o 255 (preto). Num desenho feito com esta técnica, uma marca d'água verdadeira é obtida ao enegrecer muito ligeiramente as regiões do desenho onde se quisesse que aparecesse a tal da marca d'água. Tal enegrecimento poderia ser obtido ao se subtrair um valor pequeno v, por exemplo $v = 10$, de cada ponto cuja cor pertencesse à região da marca d'água. (Naturalmente que os resultados seriam ajustados de forma a não se permitirem resultados negativos.) Se este valor pequeno fosse $v = 1$, a marca d'água era quase imperceptível devido à sutileza da diferença entre os dois níveis de cinza. Assim foi decidido que os b bits menos significativos (tipicamente, $b = 1$) de cada tom de cinza de cada ponto do desenho seriam reservados para codificar o texto a ser ocultado. No caso de uma marca d'água falsa, que oculta um texto T, diversos pontos do desenho original são escolhidos para receber a marca d'água falsa e têm seus b bits menos significativos alterados. Diferentemente da marca d'água verdadeira, a estes b bits menos significativos do ponto do desenho, são somados b bits provenientes do texto T e os $8 - b$ bits mais significativos do ponto do desenho não são alterados de forma alguma. Descrevemos a seguir a técnica mais formalmente.

Sejam inteiros positivos $m > 0$ e $n > 0$ e seja um desenho representado por uma matriz D de dimensões $m \times n$. Cada elemento da matriz é um inteiro que codifica um tom de cinza (que varia de 0 para preto a 255 para branco). Seja T um texto com $k \geq 0$ letras, onde cada letra é representada

por um byte (8 bits)[1]. Queremos obter uma matriz D', de dimensões $m \times n$, que represente o desenho com a marca d'água falsa que oculta o texto T. Sejam os inteiros b e d parâmetros arbitrariamente escolhidos. Um ponto do desenho com marca d'água falsa será reservado para portar informação relativa aos parâmetros b e d. Se por um lado o texto for muito comprido comparado com o número de pontos do desenho (o que equivale a dizer que $8k > mn - 1$), mais de um bit menos significativo por ponto deve ser usado para codificar o texto T. Assim o parâmetro b designa o *número de bits menos significativos* a ser usado no desenho com marca d'água falsa que oculta T e deve satisfazer às seguintes restrições:

$$(mn-1)b \geq 8k \tag{1}$$

$$b = 0 \text{ ou } b \text{ é divisor positivo de } 8. \tag{2}$$

Se por outro lado o texto for bastante pequeno face às dimensões da matriz, nem todos os pontos do desenho receberão marca d'água falsa. É possível, no caso, espaçar os pontos que receberão a marca d'água falsa. Assim, define-se d como o *espaçamento entre linhas e colunas* dos pontos do desenho que recebem a marca d'água falsa e as coordenadas dos pontos de D que receberão a marca d'água falsa são, na ordem:

$$\begin{array}{llll}
(d-1, d-1), & (d-1, 2d-1), & \ldots, & (d-1, \lfloor \frac{n}{d} \rfloor d - 1), \\
(2d-1, d-1), & (2d-1, 2d-1), & \ldots, & (2d-1, \lfloor \frac{n}{d} \rfloor d - 1), \\
\vdots & \vdots & \ddots & \vdots \\
(\lfloor \frac{m}{d} \rfloor d - 1, d-1), & (\lfloor \frac{m}{d} \rfloor d - 1, 2d-1), & \ldots, & (\lfloor \frac{m}{d} \rfloor d - 1, \lfloor \frac{n}{d} \rfloor d - 1).
\end{array}$$

Observe-se que teremos[2] $\lfloor \frac{m}{d} \rfloor \lfloor \frac{n}{d} \rfloor$ pontos do desenho, cada um deles ocultando b bits de informação em seus b bits menos significativos, o que resulta em $\lfloor \frac{m}{d} \rfloor \lfloor \frac{n}{d} \rfloor b$ bits disponíveis para ocultação. Para permitir a identificação não ambígua dos parâmetros b e d durante a fase de decodificação, os b bits menos significativos da posição $(d-1, d-1)$ do desenho deverão ocultar o valor[3] de b. Assim, precisaremos ocultar um total de $8k + b$ bits. Se observarmos que

$$\lfloor \frac{m}{d} \rfloor \lfloor \frac{n}{d} \rfloor b \leq \frac{mn}{d^2} b,$$

resulta então que d deve satisfazer às desigualdades abaixo:

$$\lfloor \frac{m}{d} \rfloor \lfloor \frac{n}{d} \rfloor b \geq 8k + b \tag{3}$$

$$d \leq \lfloor \sqrt{\frac{bmn}{8k+b}} \rfloor \tag{4}$$

$$1 \leq d \leq \min(m, n) \tag{5}$$

Quanto aos bits de T a serem ocultados, são considerados primeiramente os b bits menos significativos da letra $T[0]$, depois os b bits seguintes, até que finalmente tenhamos ocultado os

[1] A 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 isolatin1, que codificam letras acentuadas nos números de 128 a 255.

[2] Relembramos que, dado x um real qualquer, $\lfloor x \rfloor$ denota o `maior inteiro menor ou igual a` x. Assim, $\lfloor \frac{m}{d} \rfloor$ denota o quociente inteiro da divisão de m por d. Em C, isto coincide com a divisão de duas variáveis inteiras não negativas.

[3] Certamente, b pode ser representado com b bits.

b bits mais significativos de $T[k-1]$. Para permitir a identificação não ambígua dos parâmetros b e d durante a fase de decodificação, os bits de T a serem codificados devem ser precedidos do número b, que certamente pode ser representado com b bits. Usando a suposição de que b é um divisor de 8, a seqüência de números de b bits a ser ocultada nas falsas marcas d'água é[4]:

$$
\begin{array}{llll}
b, & & & \\
\frac{T[0]}{2^0} \bmod 2^b, & \frac{T[0]}{2^b} \bmod 2^b, & \ldots, & \frac{T[0]}{2^{8-b}} \bmod 2^b, \\
\frac{T[1]}{2^0} \bmod 2^b, & \frac{T[1]}{2^b} \bmod 2^b, & \ldots, & \frac{T[1]}{2^{8-b}} \bmod 2^b, \\
\vdots & \vdots & \ddots & \vdots \\
\frac{T[k-1]}{2^0} \bmod 2^b, & \frac{T[k-1]}{2^b} \bmod 2^b, & \ldots, & \frac{T[k-1]}{2^{8-b}} \bmod 2^b.
\end{array}
$$

Observe que se x é o tamanho desta seqüência, os $\lfloor md \rfloor \lfloor nd \rfloor - x$ últimos pontos de D escalados para receber a marca d'água falsa permanecerão inalterados.

Daremos um exemplo da ocultação de b bits propriamente dita. Sejam $x, z \in \{0, 1, \ldots, 255\}$ e $y \in \{0, 1, \ldots, 2^b - 1\}$ números inteiros tais que:

- x é um tom de cinza de um ponto a receber marca d'água no desenho D;
- y é o número inteiro formado por b bits de T e;
- z é um tom de cinza de um ponto no desenho D' que recebeu uma marca d'água.

Observe que, dados dois valores entre x, y e z, sempre é possível determinar o terceiro. Em particular, pode-se verificar que

$$z = \lfloor \frac{x}{2^b} \rfloor 2^b + ((x + y) \bmod 2^b) \tag{6}$$

$$y = (z - x + 256) \bmod 2^b. \tag{7}$$

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 exemplo a seguir.

[4] mod (`%` em C) é o operador binário que calcula o resto da divisão inteira.

```
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 o maior número 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 "negro" 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
```

3 Detalhamento do EP

Neste EP vocês devem implementar as funções abaixo, como descrito em suas definições. Você pode, se desejar, implementar outras funções para resolver o EP.

Adote

```
#define MAX        800
#define MAX2    640000
#define FNMAX      200
#define cinza      int
```

1) Escreva em C uma função LeDesenho de protótipo

```
int LeDesenho( char nomearq[FNMAX], cinza 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 qualquer erro, e deve devolver 1

caso algum erro tenha sido encontrado (ou por conta da manipulação do arquivo, ou por conta do 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 erro algum, e deve devolver 1 caso algum erro tenha sido encontrado (ou por conta da manipulação do arquivo, ou por conta do valor de `MAX2` ser insuficiente).

3) Escreva em C uma função `BeDeu` de protótipo

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

A função deve calcular os parâmetros do processo de ocultação por marcas d'água de um texto de `k` caracteres através de um desenho de dimensões `m` por `n`. O número de bits menos significativos b a ser usado no desenho com marca d'água é devolvido em `*pb`. Dentre as diversas opções possíveis que satisfaçam as equações (2) e (1), deve-se escolher aquela que minimiza b de forma que o desenho com marca d'água falsa fique o mais fiel possível ao desenho original[5]. O espaçamento entre linhas e colunas d é devolvido em `*pd`. Para tornar a marca d'água o mais imperceptível possível, busca-se espalhar ao máximo os pontos com marca d'água. Assim, uma vez escolhido b mínimo, deve-se escolher d máximo dentre aqueles que satisfazem às equações (3) e (5). Não é difícil verificar que um tal d máximo satisfará à equação (4) e será relativamente próximo ao limitante superior dado por esta equação[6]. A função `BeDeu` deve devolver 0, se for possível encontrar parâmetros que satisfaçam as restrições necessárias e deve devolver 1 caso não seja possível.

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

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 próximos[7] 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`.

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`.

[5] Observe que $b = 0$ se e só se $k = 0$.

[6] O uso desta informação pode acelerar o cálculo de d mas não é necessário.

[7] Se b não fosse um divisor de 8, poderia ocorrer que a 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]`.

m	n	k	b	d
7	7	0	0	7
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 `BeDeu`

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( cinza D[MAX][MAX], int m, int n,
               char T[MAX2], int k,
               cinza Dl[MAX][MAX], int b, int d,
               int modo);
```

A função recebe um desenho representado pela matriz `D`, de dimensões `m` por `n`, recebe um texto `T` de `k` letras, 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 calcular um desenho com marca d'água falsa que codifica `T` e armazená-lo na matriz `Dl`, 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

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

A função recebe o desenho representado pela matriz `D`, de dimensões `m` por `n`, e que devolve o máximo da matriz.

7) Escreva em C uma função `EscreveDesenho` de protótipo

```
int EscreveDesenho( char nomearq[FNMAX], cinza 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 erro algum e 1 caso algum erro tenha sido encontrado por conta da manipulação do arquivo.

8) Escreva em C uma função `DeBeDeu` de protótipo

```
void DeBeDeu( cinza D[MAX][MAX], cinza Dl[MAX][MAX],
              int m, int n, int *pb, int *pd);
```

A função recebe dois desenhos representados pelas matrizes `D` e `Dl`, ambas de tamanho `m` por `n`. Sabe-se que `Dl` foi obtida de `D` através de uma ocultação de texto por marca d'água. A função deve detectar o espaçamento entre linhas e colunas d que foi usado na ocultação e devolvê-lo em `*pd`. Da mesma forma, a função deve detectar o número de bits menos significativos b que foi usado no processo e devolvê-lo em `*pb`. Sugestão: calcule o menor i para o qual a diferença `Dl[i][i]-D[i][i]` não seja nula.

9) Escreva em C uma função `DeCodifica` de protótipo

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

`DeCodifica` recebe na matriz `D`, de dimensões `m` por `n`, um desenho que sofreu alterações por um processo de ocultação de texto por marca d'água de forma a obter o desenho recebido na matriz `Dl`, também de dimensões `m` por `n`. Os parâmetros b e d usados no processo de ocultação são recebidos em `b` e em `d`, respectivamente. Se o modo de operação recebido em `modo` for verborrágico, a função deve imprimir alguns dados (descritos mais adiante). Esta função deve devolver em `T` o texto que foi codificado e deve devolver k, o número de letras que foram originalmente codificadas. OBS: para evitar ambiguidades, podemos supor que o texto ocultado não termina com o caractere `'\0'`.

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` letras. A função deve devolver 0 se não houver qualquer 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 seguintes operações abaixo descritas e as executa. Isto deve ser feito repetidamente até que a operação 0 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' com marca d'água falsa.

Este desenho D' é o desenho obtido a partir de D através da ocultação do texto T por marca d'água. No arquivo em formato PGM que contém D', o valor do máximo valor de um tom de cinza deve ser o máximo entre: o valor correspondente no arquivo PGM que foi lido e contém o desenho D; e o máximo inteiro da matriz D'. Observe que os desenhos D e D' serão idênticos se e só o texto T não tiver letra alguma.

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' com marca d'água falsa;
3. o nome de um arquivo que deverá conter um texto T obtido da decodificação de D e D'.

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') 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, os mesmos deverão ser impressos, nesta ordem;

3. Toda vez que um texto T for lido com sucesso de um arquivo ou decodificado, o número k de suas letras 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 forem extraídos alguns bits de texto a partir de um ponto que recebeu uma marca d'água**:

1. A posição do desenho escolhida que recebeu a marca d'água;

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 a marca d'água falsa.

As impressões dos tons de cinza devem ser feitas com formato `%02x`, para impressão de inteiro em formato hexadecimal[8].

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

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