CEFET/RJ

Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

---

Disciplina de Organização e Estrutura de Arquivos

Estudo e Manipulação de Imagens Bitmap

Marcos Eduardo Carvalho Teixeira

Dezembro, 2017

Objetivo/Resumo. Análise e manipulação de imagens ***Bitmap (24-bit - sem compressão)*** para compreensão da estrutura de arquivo presente nesse formato de arquivo, assim como estudo e experiência com o método de acesso aleatório de arquivos ("Random Access File").

1. Introdução

O trabalho apresenta códigos desenvolvidos para realizar a manipulação de imagens "***.BMP"*** (Bitmap). As funcionalidades desenvolvidas foram o Espelhamento (Eixo-X — Horizontal), a Inversão (Eixo-Y — Vertical), a alteração para Escala Cinza, a Rotação-90° e a Dobra das Dimensões (Altura e Largura).

2. Desenvolvimento/Descrição

Todo código foi implementado na linguagem Java (SE-1.8), utilizando a IDE Eclipse Java Oxygen. Para leitura e escrita dos arquivos Bitmap, foi utilizada a classe RandomAccessFile (*java.io.RandomAccessFile*), pois esta permite encarar o arquivo como um vetor de bytes e fazer acessos/alterações pontuais, sem a necessidade de percorrer o arquivo sequencialmente.

Alguns métodos básicos da classe RandomAccessFile foram utilizados em todas as funcionalidades, eles são:

".seek" — Define um ponteiro para uma posição específica no arquivo.

".read" — Lê os bytes a partir do ponto especificado pelo método 'seek' e retorna um vetor de bytes de tamanho igual ao enviado como parâmetro.

".getFilePointer" — Retorna um valor *long* que representa a posição do ponteiro até aquele momento na execução.

Para focar o estudo e trabalho, algumas "simplificações" foram feitas, como o não tratamento direto das exceções EOFException e IOException. Além disso, o padrão de Bitmap analisado e trabalhado foi o 'Bmp 24-bit RGB', com tamanho de cabeçalho (*header*) de 54 bytes, sem palheta de cores e sem compressão.

O formato abordado possui o cabeçalho (54 bytes) dividido em 14 bytes de cabeçalho-de-arquivo e 40 bytes de cabeçalho-de-imagem. Este é subdividido em onze grupos de bytes, dentre eles a largura e altura da imagem (3 bytes cada), que foram utilizados em todas as funcionalidades. Um exemplo da estrutura é mostrado na Fig. 1, os bytes são representados por números em hexadecimal no intervalo de 00 até FF (0 - 255 em decimal).

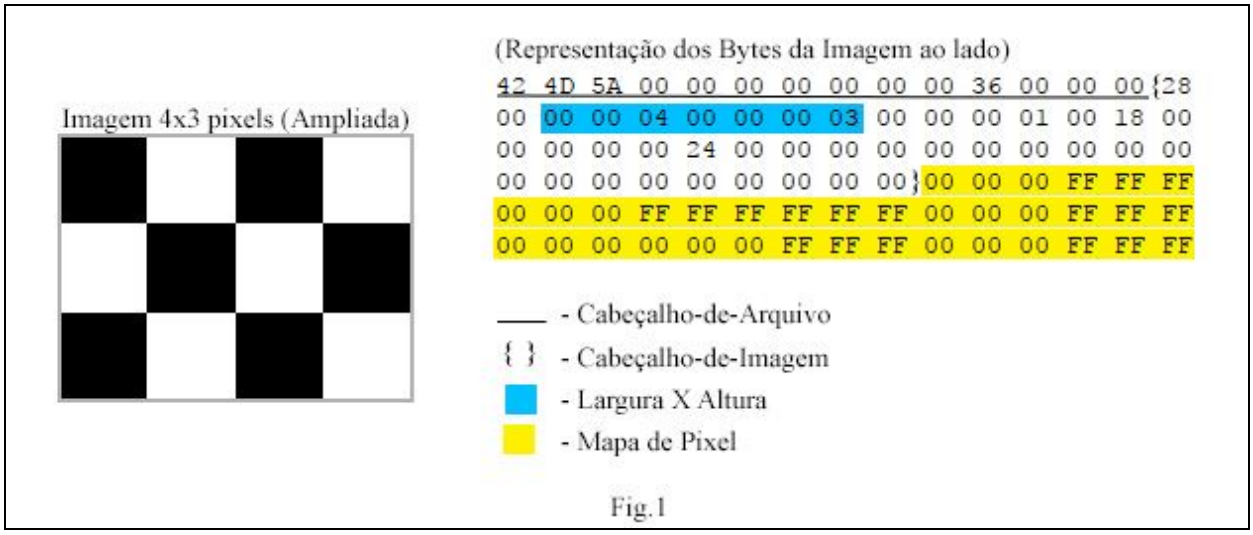
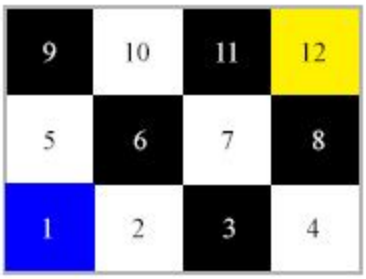


Fig.1

Na Fig.1 também está ilustrado o mapa de pixels, que no formato em questão representa explicitamente cada pixel e sua posição. Cada pixel é formado por três bytes que formam uma cor sólida no padrão RGB, porém, no caso da imagem bitmap, sua leitura é feita ao contrário, sendo o primeiro byte para a proporção de azul, o segundo para verde e terceiro vermelho. Por exemplo, os bytes 00 FF FF representam a soma do vermelho total - FF, com o verde total - FF e azul nulo - 00, gerando a cor amarela.

O mapa de pixels é processado de acordo com a altura e largura contida no cabeçalho, cada sequência de pixels, com tamanho equivalente à largura, é interpretada como uma linha de uma matriz, e portanto, o número de linhas será igual à altura da imagem.

O mapa está expresso sequencialmente, interpretando-o como uma matriz, a primeira linha de pixels da imagem corresponde a última do arquivo . Exemplo na Fig.2.



1	2	3	4
FF 00 00	FF FF FF	00 00 00	FF FF FF
5	6	7	8
FF FF FF	00 00 00	FF FF FF	00 00 00
9	10	11	12
00 00 00	FF FF FF	00 00 00	00 FF FF

Fig.2

O código inicia declarando duas instâncias da classe RandomAccessFile, uma para a imagem original e outra para a final. Além disso, faz a leitura (seek & read) das dimensões da imagem

(de acordo com a posição dos bytes de maneira explícita) para serem passadas como parâmetro para os métodos. Em sequência, lê e escreve o cabeçalho de uma imagem para a outra.

```
Scanner scanMenu= new Scanner(System.in);
System.out.print("Digite o diretorio + nome do arquivo (Ex.: files/ttt.bmp):");
String fileS = scanMenu.nextLine();
RandomAccessFile img =  new RandomAccessFile(fileS, "rw");
RandomAccessFile imgF =  new RandomAccessFile(fileS + "-new.bmp", "rw");

img.seek(0x12);

byte [] width = new byte [4];
byte [] height = new byte [4];
img.read(width);
img.read(height);

int headLgt= 0x36;
img.seek(0);
byte [] head = new byte [headLgt];
img.read(head);

imgF.write(head);

width = inverte(width);
height = inverte(height);

int w = ByteBuffer.wrap(width).getInt();
int h = ByteBuffer.wrap(height).getInt();
int wBytes= w*3;
```

A altura e largura da imagem no cabeçalho está representada em hexadecimal, com o número de menos significativo no último byte (00 00 04 x 00 00 03). Para contornar essa situação foi criado o método “Inverte”, que recebe um vetor de bytes, previamente obtido pelo método “read”, e o retorna invertido, como no exemplo abaixo.

Recebe:

2C	**00**	**FF**

Retorna:

FF	**00**	**2C**

```java
public static byte[] inverte(byte [] a) {

    int j= a.length;
    byte aux[] = new byte[a.length];
    j--;
    for (int i = 0; i < a.length; i++) {
        aux[j] = a[i];
        j--;
    }
    return aux;
}
```

Método Inverte

Define um vetor auxiliar (AUX) que vai receber os valores do fim até o início do vetor "a".

Espelhamento (Eixo-X — Horizontal). Gera uma nova imagem espelhada no eixo horizontal com base na imagem alvo. Para cada linha de altura do mapa, é repetido o método "invertePixel" , que realiza o mesmo processo do método inverte, porém, de três em três bytes.

```java
public static byte[] invertePixel(byte [] a) {
    int j= a.length;
    byte aux[] = new byte[a.length];
    j--;
    for (int i = 0; i < a.length;) {
        aux[j-2] = a[i];
        aux[j-1] = a[i+1];
        aux[j] = a[i+2];
        j-= 3;
        i+=3;
    }

    return aux;
}

public static RandomAccessFile espelhar(RandomAccessFile img,
        RandomAccessFile imgF, int wBytes, int h, int head) throws Exception{
    byte [] aux = new byte [wBytes];
    img.seek(head);
    for(int i = 0; i<h; i++) {
        img.read(aux);

        aux = invertePixel(aux);

        imgF.write(aux);

    }
    return imgF;
}
```

Inversão (Eixo-Y — Vertical). Gera uma nova imagem invertida no eixo Y de mesmas dimensões, ou seja, 'de cabeça para baixo'. Lê, inverte (invertePixel) e escreve a última linha da imagem original, como a primeira da nova imagem e segue sucessivamente até a chegar no cabeçalho.

```
public static RandomAccessFile inverter(RandomAccessFile img,
        RandomAccessFile imgF, int wBytes, int h) throws Exception{
    byte [] aux = new byte [wBytes];
    for(int i = 0; i<h; i++) {
        int l = i + 1;
        img.seek(img.length() - l*wBytes);
        img.read(aux);
        aux = invertePixel(aux);
        imgF.write(aux);
    }
    return imgF;
}
```

Escala Cinza. Gera uma nova imagem a partir da original em "preto-e-branco", ou melhor, altera cada pixel da imagem original para um equivalente na escala de tons de cinza.

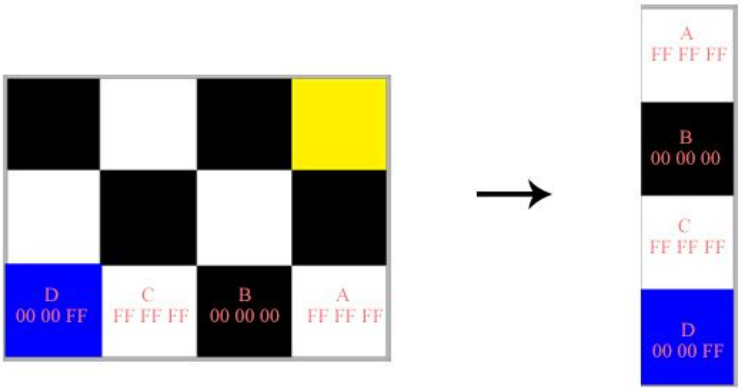
O método multiplica cada byte dos pixels por uma diferente constante cada, faz a soma dos resultados e gera um valor único que é atribuído às três posições RGB. Este processo é repetido por todas as linhas da imagem.

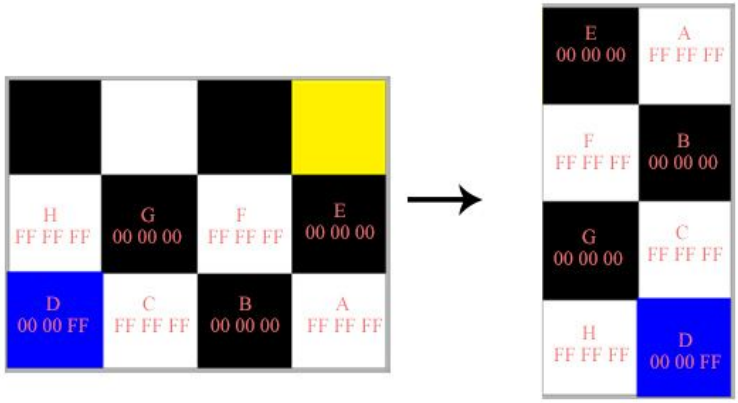
```
public static RandomAccessFile escalaCinza(RandomAccessFile img,
        RandomAccessFile imgF, int wBytes, int h, int head) throws Exception{
    byte [] aux = new byte [wBytes];
    img.seek(head);
    for(int i = 0; i<h; i++) {
        img.read(aux);
        for(int j= 0; j<aux.length;) {
        int a = (int) aux[j];
        int b = (int) aux[j+1];
        int c = (int) aux[j+2];

        int m= (int) ((0.58*a) + (0.17*b) + (0.8*c));

        aux[j] = (byte) m;
        aux[j+1] = (byte) m;
        aux[j+2] = (byte) m;

        j+=3;
        }
        imgF.write(aux);
    }
    return imgF;
}
```

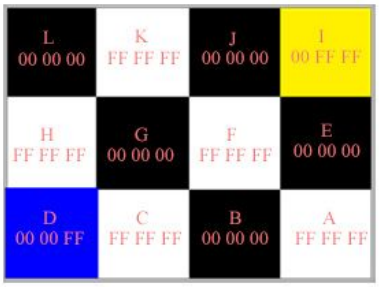
Rotação-90°. É gerada uma nova imagem com dimensões trocadas e com todos os pixels rotacionados para uma posição equivalente a 90°. Esta função realiza alterações no cabeçalho da imagem, ela troca o valor da altura pelo da largura e vice-versa.

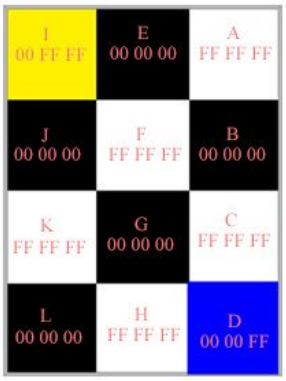
A imagem é encarada como uma matriz, a quantidade de linhas é igual a altura e a de colunas é a quantidade de pixels por linhas, porém, cada pixel equivale a 3 bytes. O método realiza uma transposição dessa matriz.

Para a transposição, todos os pixels da última linha da imagem original, começando do final, são postos cada um na última posição de cada linha da imagem nova. Assim, todos os pixels da penúltima linha são colocados na penúltima posição de cada linha da nova imagem. O processo de rotação é ilustrado a seguir.










```
public static RandomAccessFile rotacionar(RandomAccessFile img, RandomAccessFile imgF,
        int h, int header, int w, byte[] width, byte[] height) throws Exception{
    int newBWd = h*3;
    byte [] aux = new byte [newBWd];

    int pointer = 0;
    for(int i= 0; i<w; i++) {
        int k =0;
        for(int j=h; j>0; j-- ) {
            pointer = header + (j-1)*(w*3);
            pointer += i*3;
            img.seek(pointer);

            aux[k] = img.readByte();

            aux[k+1] = img.readByte();

            aux[k+2] = img.readByte();

            k+=3;
        }

        imgF.seek(imgF.length());
        imgF.write(aux);
    }

    imgF.seek(0x12);

    imgF.write(inverte(height));
    imgF.seek(0x16);
    imgF.write(inverte(width));
    return imgF;

    }
```

Dobra das Dimensões. Gera uma nova imagem em que a altura e largura tem seu valor dobrado. Para multiplicar as dimensões por dois, é necessário converter as dimensões que estão em um

vetor de bytes para um único número inteiro. Para este propósito, foram utilizados os seguintes métodos, na seguinte ordem:

“ByteBuffer.wrap().getInt()” — Para fazer a conversão de byte para inteiro.
“Integer.toHexString” — Converter um inteiro multiplicado por dois para a forma hexadecimal em uma nova *String*.
“Long.parseLong” — Converte a String de Hexadecimal para um valor long.
“ByteBuffer.allocate().putInt()” — Retorna o valor de long para um vetor de byte.

Após a alterar as dimensões, o método escreve cada linha e pixel do mapa original duas vezes na nova imagem.

```
public static RandomAccessFile dobrar(RandomAccessFile img, RandomAccessFile imgF,
        int h, int header, int w, byte[] width, byte[] height) throws Exception{

        int a = ByteBuffer.wrap(width).getInt();
        String aS = Integer.toHexString(a*2);
        a = (int) Long.parseLong(aS, 16);
        ByteBuffer wBB = ByteBuffer.allocate(4).putInt(a);
        width = wBB.array();
        int b = ByteBuffer.wrap(height).getInt();
        String bS = Integer.toHexString(b*2);
        b = (int) Long.parseLong(bS, 16);
        ByteBuffer hBB = ByteBuffer.allocate(4).putInt(b);
        height = hBB.array();

        imgF.seek(0x12);
        imgF.write(inverte(width));
        imgF.seek(0x16);
        imgF.write(inverte(height));
        imgF.seek(header);
        long position = header;
        byte [] aux = new byte [w*3];
        byte [] auxD =  new byte [(aux.length)*2];
        for(int i= 0; i<h; i++){
            img.seek(position);
            img.read(aux);
            int k=0;
            for (int j = 0; j < aux.length;) {
                auxD[k] = aux[j];
                auxD[k+1] = aux[j+1];
                auxD[k+2] = aux[j+2];
                auxD[k+3] = aux[j];
                auxD[k+4] = aux[j+1];
                auxD[k+5] = aux[j+2];

                j+=3;
                k+=6;
                }

            imgF.write(auxD);
            imgF.write(auxD);
            position = img.getFilePointer();
        }
        return imgF;
        }
```

3. Testes/Experimentos

O código foi compilado pela IDE e executado em console. Os testes foram realizados em uma máquina com a configuração: 8 gb de RAM, processador i5 e Windows 10 64 bits.

Duas imagens diferentes foram usadas para testar cada função :

Barn.bmp

cefet.bmp

Nas funções Espelhamento e Inversão, o processamento das duas imagens ocorreu como o esperado, retornado em ambos os casos a imagem esperada.

Espelhamento

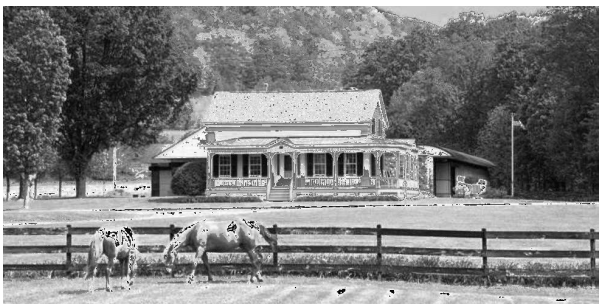
	

Inversão

O método Escala Cinza obteve resultado próximo ao esperado. A maior parte dos pixels obteve um tom de cinza coerente com a cor original, porém, alguns acabaram totalmente pretos ou brancos.

Escala Cinza



A rotação apresentou resultados inconclusivos, ocorrendo como esperado somente na segunda imagem. As dimensões da primeira foram trocadas com sucesso, mas a transposição de pixels não correu corretamente.

Rotação

4. Conclusão

Mesmo que alguns testes não tenham sido bem sucedidos, com o desenvolvimento das funcionalidades, se cumpriu o objetivo do trabalho.

Depois de uma série análises e comparação entre arquivos, chegou-se à conclusão que as imagens Bitmap apresentam uma estrutura de arquivo simples e de fácil manipulação.

O método de acesso aleatório aos bytes de arquivos, por conta da precisão na leitura e escrita de bytes, foi indispensável para as operações desenvolvidas.