Compressão de Imagens

1 Considerações Iniciais

Neste trabalho, você deverá implementar em Linguagem C um algoritmo para compressão de imagens baseado em uma decomposição de blocos quadrados. A Figura 1 ilustra o funcionamento desse algoritmo para uma imagem de entrada **I**. A imagem **R** foi reconstruída após o processo de compressão.

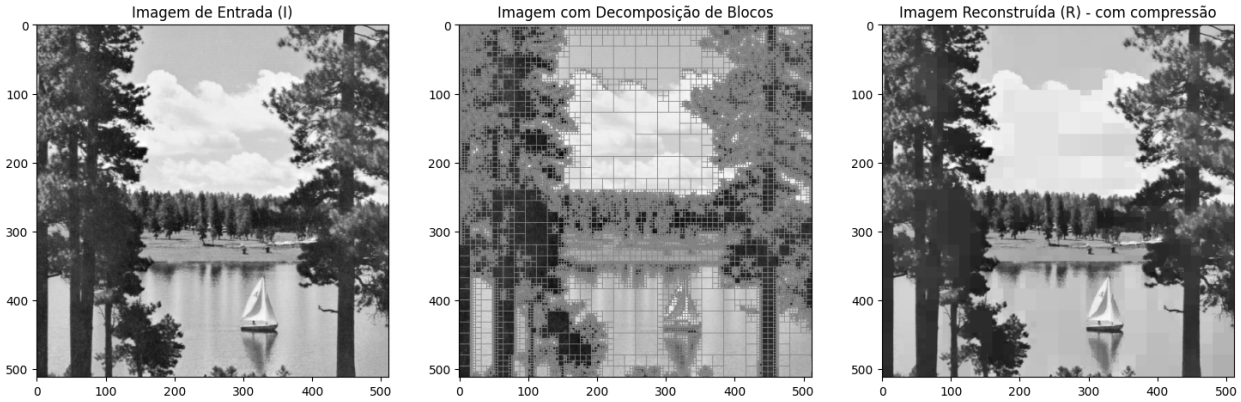

Figura 1: O processo de compressão envolve dividir a imagem de entrada **I** em blocos quadrados seguindo um critério de homogeneidade. Logo após, utilizar as informações resultantes dessa divisão para reconstruir a imagem de entrada, gerando a imagem **R**.

Sugestão para iniciar uma pesquisa na área, consulte:

UNGUREANU, Vlad-Ilie; NEGIRLA, Paul; KORODI, Adrian. Image-Compression Techniques: Classical and "Region-of-Interest-Based" Approaches Presented in Recent Papers. Sensors, v. 24, n. 3, p. 791, 2024.

2 Algoritmo de Compressão

Em um algoritmo de compressão de dados, o codificador é a parte responsável por transformar a entrada, como uma imagem, em uma representação comprimida de forma eficiente, visando reduzir o tamanho dos dados. O codificador desempenha um papel essencial no processo de compressão, pois ele deve minimizar a redundância e eliminar informações desnecessárias, preservando ao máximo a qualidade. Vamos chamar essa representação comprimida da imagem de entrada de *bitstream* **B**. Portanto, podemos definir como:

$$\mathbf{B} = C(\mathbf{I}), \tag{1}$$

onde **I** corresponde a imagem de entrada e C o algoritmo de compressão a ser empregado.

O *bitstream* deve reunir todas as informações necessárias para que o *decodificador* D possa reconstruir a imagem de entrada. Portanto,

$$\mathbf{R} = D(\mathbf{B}), \tag{2}$$

onde **R** representa a imagem reconstruída.

Um algoritmo de compressão eficiente irá permitir imagens reconstruídas com qualidade, ou seja, baixa percepção visual das diferenças entre **I** e **R**, e com o menor tamanho do *bitstream*.

Neste trabalho, o codificador e o decodificador devem ser construídos seguindo o fluxograma descrito na Figura 2.

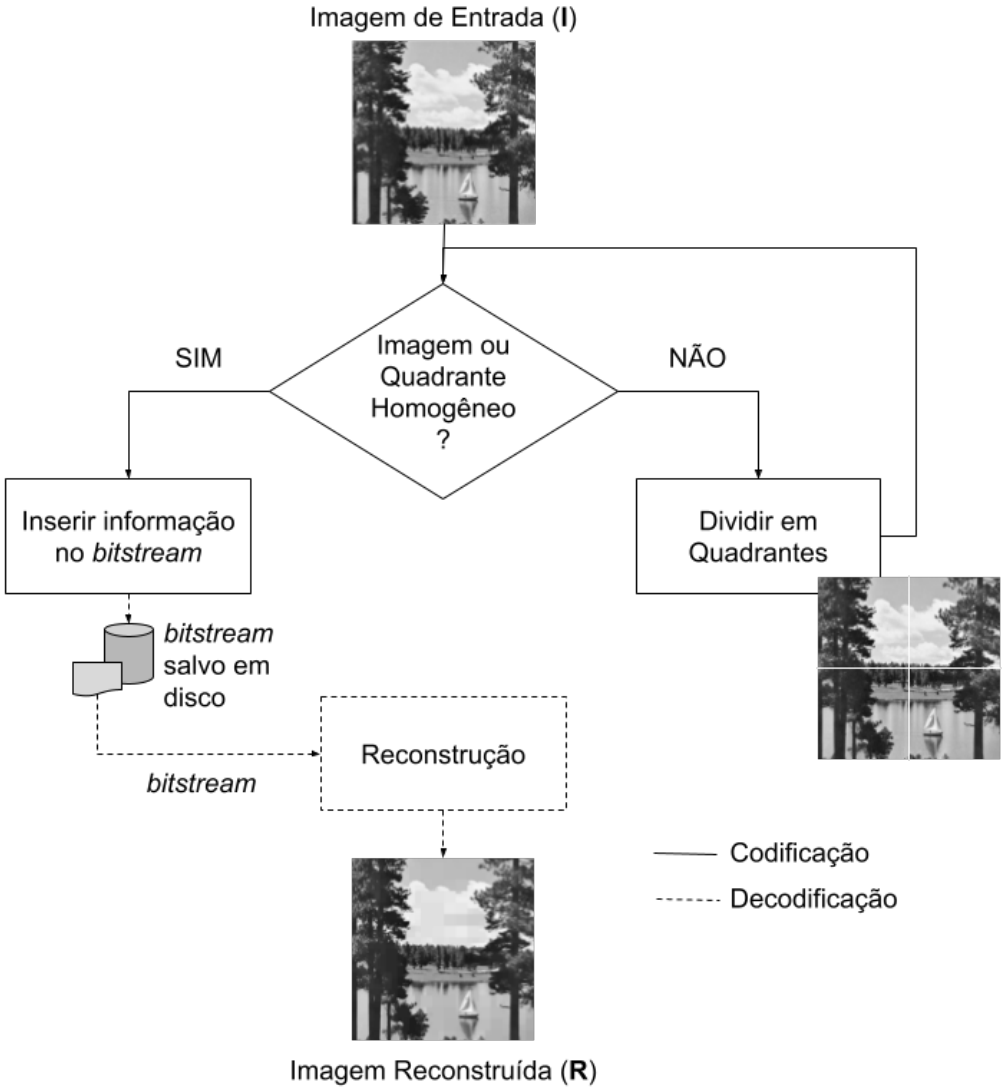

Figura 2: Procedimento realizado pelo algoritmo de compressão de imagens objeto deste trabalho. A imagem original ou os quadrantes (nas iterações) são divididos ou não conforme um critério de homogeneidade. Caso a imagem ou um quadrante não seja dividido, informações sobre a imagem ou o quadrante devem ser inseridas em um *bitstream*. O decodificador é capaz de reconstruir uma versão aproximada da imagem de entrada apenas lendo o *bitstream*.

Observações:

- Utilize imagens quadradas e no formato PGM disponíveis no diretório do projeto no DropBox.
- Você é livre para escolher o critério de homogeneidade que desejar. Pesquise sobre as medidas que podem ser utilizadas para computar a homogeneidade de um bloco de pixels de uma imagem.

- O codificador e decodificador devem ser programas diferentes.
- Utilize a função custo especificada pelo professor para relacionar qualidade da compressão e tamanho do *bitstream* gerado. O valor "custo" deve ser inserido em uma planilha disponível nesta atividade no classroom para uma imagem a ser definida pelo professor.
- A equipe deve apresentar, juntamente com códigos, os resultados de compressão: *bitstream* e imagem reconstruída.

Nas seções seguintes deste documento estão descritas algumas informações necessárias para o correto desenvolvimento deste trabalho.

3 Formato PGM

Neste trabalho, deve-se utilizar o formato PGM (*portable graymap*) para armazenar imagens em arquivos. Este formato tem duas variações, uma binária (o PGM "normal" ou *raw*) e outra textual (o PGM ASCII ou *plain*). Em ambos os casos, o arquivo deve conter um cabeçalho e a matriz correspondente à imagem. O exemplo a seguir mostra um arquivo PGM textual:

```
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 obrigatoriamente uma palavra-chave, que deve ser "P2" no caso de um arquivo PGM textual e "P5" no caso de um arquivo PGM binário. A segunda linha contém dois números inteiros que indicam o número de colunas e o número de linhas da matriz, respectivamente. A terceira linha contém um número inteiro positivo *maxval*, que deve ser igual ao maior elemento da matriz. Na definição do formato PGM, *maxval* não pode ser maior que 65535. Para fins deste trabalho, entretanto, *maxval* é no máximo 255. Os demais números do arquivo são os elementos de uma matriz de inteiros com os tons de cinza de cada ponto da imagem. 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. Todo o texto que vai desde um caractere '#' até (e inclusive) o próximo fim de linha é um comentário e deve ser ignorado. Este é um exemplo de arquivo PGM textual com um comentário:

```
P2
# feep.pgm
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 3 3 3 0 0 7 7 7 7 0 0 11 11 11 0 0 15 15 15 0 0 0
0 0 0 3 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 0 0 3 0 0 0 7 7 7 0 0 0 11 0 0 0 0 0 15 15 15 0 0
0 0 0 3 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 0 3 3 3 0 0 7 0 0 0 0 0 11 11 11 0 0 15 15 15 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

O formato PGM binário tem cabeçalho análogo ao do PGM textual, usando a palavra chave "P5" em vez da "P2". O que muda é o modo como é armazenada a matriz de tons de cinza. No formato PGM textual, essa matriz é guardada como uma sequência de caracteres ASCII contendo as representações decimais das tonalidades de cinza. No formato PGM binário, a matriz é guardada como uma sequência de *bytes*, sendo o valor de cada *byte* (de 0 a 255) uma tonalidade de cinza. O número de *bytes* da sequência é exatamente igual ao número de elementos da matriz[1]. Este é um exemplo de arquivo PGM binário:

```
P5
# feep.pgm
24 7
15
```

... (sequência de 24 x 7 bytes com as tonalidades de cinza)

Existem dois outros formatos de arquivo muito semelhantes ao PGM: o PBM (*portable bitmap*), para imagens monocromáticas (só preto e branco, sem tons de cinza), e o PPM (*portable pixmap*), para imagens coloridas. No primeiro, os elementos da matriz podem assumir apenas os valores 0 e 1. No segundo, os elementos da matriz são triplas de números inteiros positivos correspondentes às intensidades das cores vermelha, verde e azul nos pontos da imagem. Quem quiser saber mais detalhes sobre esses formatos, visite as seguintes páginas:

- http://en.wikipedia.org/wiki/Portable pixmap
- http://netpbm.sourceforge.net/doc/pbm.html
- http://netpbm.sourceforge.net/doc/pgm.html
- http://netpbm.sourceforge.net/doc/ppm.html

4 Sobre a organização dos códigos fontes

ATENÇÃO:

1. A nota máxima deste trabalho é 10.0. A distribuição dos pontos segue como definido a seguir:

   - **Entrada e saída com arquivos texto ou binário:** Pontuação: 1.0
   - **Implementação do algoritmo para divisão dos blocos conforme critério de homogeneidade:** Pontuação: 3.0
   - **Criação do *bitstream*:** Pontuação: 2.0
   - **Decodificação** : 4.0

   Critérios para cada tópico acima:

   - Organização dos códigos: 20%
   - Domínio da solução adotada: 50%
   - Análise Técnica: 30%

2. Todos os arquivos fontes do seu programa devem conter um cabeçalho como o seguinte:

[1] Na verdade, essa descrição se aplica apenas a arquivos PGM com $maxval \leq 255$, que são considerados neste trabalho. No caso $256 \leq maxval \leq 65535$, a matriz é guardada como um sequência de pares de *bytes* e o número de *bytes* da sequência é o dobro do número de elementos da matriz.

```
/*************************************************/
/* Aluno: Joao de Souza
/* Matricula: 12345
/* Avaliacao 04: Trabalho Final                  */
/* 04.505.23 - 2024.1 - Prof. Daniel Ferreira    */
/* Compilador:...(DevC++ ou gcc) versao ...      */
/*************************************************/
```

A organização do programa deve obedecer a construção de arquivos de cabeçalhos e permitir a compilação separada. Deve-se utilizar processo de *linkedição*. Procure dividir seus códigos em arquivos sempre que necessário. Enfim, procure construir bibliotecas genéricas que proporcione posterior reuso.

3. O trabalho deverá ser realizado em **equipes com no máximo 4 (quatro) participantes**. **Nota individual.** Sugiro, portanto, que marquem seminários entre a equipe para que todos possam atingir o mesmo nível de conhecimento.

4. Sugiro que o codificador fique sob a responsabilidade de dois integrantes da equipe, enquanto, o decodificador sob a responsabilidade dos demais. No caso da equipe não possuir 4 integrantes, as atividades devem ser divididas de forma igualitária entre os membros.

5. Códigos fontes copiados (com ou sem eventuais disfarces) receberão nota **ZERO**.

6. É muito importante que seu programa tenha comentários e esteja bem indentado, ou seja, digitado de maneira a ressaltar a estrutura de subordinação dos comandos dos programas (conforme estudado em aula).

7. O professor poderá executar o programa tantas vezes quantas forem necessárias para testar os vários casos possíveis para as entradas.

8. Os códigos devem ser disponibilizados ao professor por meio de um link no GitHub. Deve ser construído um README com detalhes para a execução.

9. No dia da apresentação é **indispensável** o comparecimento de toda a equipe. Faltas devem ser devidamente justificadas para segunda-chamada.

10. Datas para as apresentações:

    - **Dia 1: 16 de Setembro de 2024 a partir das 07h40.**
    - **Dia 2: 17 de Setembro de 2024 a partir das 13h30**.

A ordem de apresentação será definida posteriormente por sorteio.