-:: Projeto 2 – Codificador de imagens binárias ::-

A captura e representação digital de imagens por câmeras e máquinas fotográficas consiste em um processo que envolve várias etapas.

A aquisição é realizada por sensores ópticos, como os CCD (*Charge-Coupled Device*) ou os CMOS (*Complementary Metal-Oxide-Semiconductor*), que captam a luz proveniente da cena após atravessam as lentes do dispositivo e a converte em sinais elétricos.

Os sinais elétricos analógicos gerados pelo sensor (os elementos sensíveis à luz no sensor) são então transformados em sinais digitais por um conversor A/D, que quantiza a intensidade da luz captada em pixels individuais. Assim, uma imagem digital é dada por conjunto de pixels (unidades mínimas de informação), cujos valores representam a cor, intensidade luminosa e posição.

A resolução de uma imagem é medida pelo número de pixels que a compõem, dada em termos de sua largura e altura em pixels. Por exemplo, uma imagem pode ter dimensões de 1920 x 1080 pixels, o que significa que possui 2073600 pixels. Quanto mais pixels, maior a resolução e detalhamento da imagem capturada.

Por sua vez, a profundidade de cor, medida em bits por pixel, refere-se à quantidade de informações disponíveis para cada pixel e determina a variedade de cores que podem ser representadas por eles na imagem. Quanto maior a profundidade de cor, maior é a gama de cores e suas nuances que podem ser reproduzidas na imagem.

Por exemplo: em uma imagem com profundidade de 1 bit será possível representar apenas duas cores por pixel, geralmente preto e branco (imagem binária). Entretanto, se a profundidade for de 8 bits, é possível representar 256 valores diferentes em cada pixel.

Os canais em uma imagem se referem às diferentes componentes de cor ou informações associadas a cada pixel, enquanto a profundidade de cor está relacionada à quantidade de bits usados para representar cada canal. Se tivermos apenas um canal de cor, a variação de valores possível naquela profundidade de bits poderá representar diferentes tons de cinza (isto é, a variação sendo expressa entre o branco e o preto). Em uma imagem RGB há três canais (vermelho, verde e azul), a cor de cada pixel é descrita por três valores que representam, respectivamente, a intensidade de componente vermelho, verde e azul que constitui aquela cor.

Todas essas informações, e as vezes até outra mais, precisam de alguma forma serem armazenadas em arquivo para que uma imagem digital seja representada. A especificação e/ou escolha do formato de arquivo para representar imagens depende de necessidades específicas, por exemplo: se é importante manter a qualidade da imagem, se é necessário ter transparência, se o tamanho do arquivo é um fator significativo etc.

O formato JPEG (*Joint Photographic Experts Group*), por exemplo, utiliza compressão com perdas para reduzir o tamanho do arquivo; enquanto o PNG (*Portable Network Graphics*), que suporta transparência por possuir um canal adicional (alfa), preserva a qualidade da imagem e geralmente resulta em arquivos maiores. Por sua vez, o formato GIF (*Graphics Interchange Format*) utiliza uma paleta de até 256 cores de três canais e representa cada pixel por um único valor correspondendo uma entrada da paleta. Isso permite uma redução na quantidade de bytes por pixel, mas pode resultar em perda de qualidade em imagens com muitas cores dado o tamanho limitado da paleta (que precisará fazer cores e nuances diferente, mas parecidas, serem representados pela mesma cor na paleta).

Considere assim que desejamos criar um formato para representar imagens binárias baseado em um sistema de codificação inovador e único. A ideia é processar uma imagem de tamanho N x M, dadas apenas por pixels pretos ou brancos, e produzir como saída um código que corresponde a uma transliteração exata da imagem e que é dada por uma sequência única e constituída pelas letras P, B e X.

O processo de codificação é essencialmente simples, sendo constituído de duas regras apenas e que são descritas a seguir:

1. Se uma imagem é uniforme, possuindo todos os pixels de uma mesma cor, o código gerado para ela será dado simplesmente pela letra correspondente a cor de seus pixels. Por exemplo, se uma imagem for toda branca, o código resultante é B (branco); e no caso a imagem seja toda preta, o código corresponderá a letra P (preto).
2. No entanto é possível que a imagem apresente pixels de cores diversificadas. Neste caso, o código emitido na saída será dado pela letra X e, dois cortes, um horizontal e outro vertical, são realizados produzindo até quatro subimagens.
   Cada corte é realizado aproximadamente no ponto médio da imagem, dividindo a imagem na metade. Após ambos os cortes serem realizados, o processo então continua com a analisa cada uma das subimagens, completando o código com o resultado do processamento do 1º, 2º, 3º e 4º quadrantes, nesta ordem.
   Obs: Dado que o pixel é a unidade indivisível da imagem, o processo de corte não deve resultar partes fracionárias ou meios pixels. Considere, nestes casos, que as subimagens do lado esquerdo resultantes do corte vertical terão uma coluna a mais do que as da direita; e de modo análogo, as subimagens superiores terão uma linha a mais do que as inferiores.

Vejamos um exemplo de como é feita a codificação de uma imagem segundo esse processo. Considere que o grid dado pela Figura 1 represente uma imagem binária de dimensões 3x5, onde os elementos escuros correspondem a pixels pretos e as que são dadas em claro como sendo pixels brancos.

Figura 1 - Imagem binária de dimensões 3x5.

Como a imagem não é homogênea é geramos um código X e realizamos os dois cortes, dividindo a imagem em quatro partes. Observe não é permitido ter “meios pixels”, então a quantidade maior de colunas ficará para a metade à esquerda; e a quantidade maior de linhas para a metade de cima. A Figura 2 ilustra os pontos de corte e as quatro subimagens resultantes do processo de divisão.

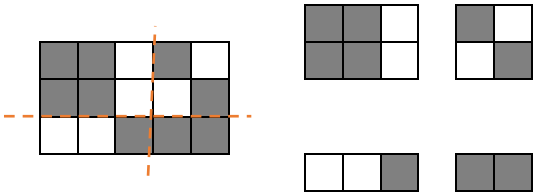


Figura 2 - Cortes horizontal e vertical na imagem resultando em quatro subimagens: superior esquerda (1º quadrante); superior direita (2º quadrante); inferior esquerda (3º quadrante) e inferior direita (4º quadrante).

O processo de codificação deve continuar analisando cada uma das subimagens resultantes segundo a ordem de seus quadrantes. O código desta imagem será dado então pela letra X seguida da concatenação dos códigos resultantes da análise de cada uma das subimagens.

Para a subimagem do 1º quadrante, que também não é homogênea, teremos a emissão de um X e a divisão em 4 partes, ficando da seguinte forma:

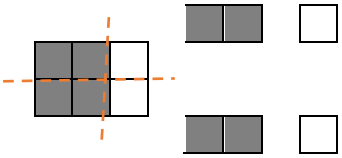


Figura 3 - Processamento da subimagem do 1º quadrante e que também resultando em quatro subimagens.

Até esse momento o código emitido corresponde a sequência "XX" e, tal qual anteriormente, devemos analisar as imagens dos quatro quadrantes em ordem. Note que a imagem do 1º quadrante contém apenas pixels pretos e o resultado da sua codificação resultará em P. Em seguida, processamos a subimagem correspondente ao 2º quadrante e cujo resultado será B, dado que é uma imagem formada apenas por um pixel branco. Para o 3º quadrante, assim como no caso do 1º, o código será P; e para o 4º quadrante, teremos um B.

Assim, até o presente instante, o código parcial para a imagem inicial será X seguido de XPBPB (do 1º quadrante) concatenado com o código resultante do processamento dos demais quadrantes (2º, 3º e 4º).

Analisemos agora a subimagem correspondente ao 2º quadrante. Ela também não é homogênea, produzindo um X e cuja divisão resultará em 4 partes (como ilustrado na Figura 4).

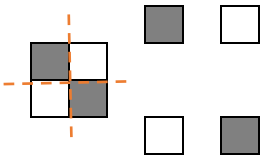


Figura 4 - Processamento do 3º quadrante da imagem original usada como exemplo.

Como resultado desse processamento teremos o código XPBBP, sendo: o X decorrente da necessidade de divisão (pixels de cores diversificadas), o P da subimagem do 1º quadrante, o B do 2º quadrante, o outro B dado em relação ao 3º quadrante e P referente ao último quadrante.

A terceira subimagem deverá ser processada em sequência, sendo esta tarefa ilustrada através na Figura 5.

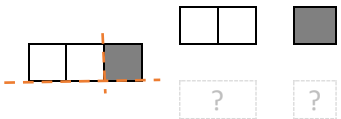


Figura 5 - A análise de uma imagem com uma linha de pixles.

Observe que em decorrência do processo de corte horizontal não poder conter pixels ao meio, neste caso não há a formação dos 3º e 4º quadrantes e o código para essa subimagem será dado apenas por XBP.

Por fim, procedemos a análise e codificação da última subimagem que corresponde ao 4º quadrante da imagem inicial, representada pela Figura 6.

Figura 6 - 4º e último quadrante da imagem inicial para processamento e codificação.

Por ser uma imagem contendo apenas pixels em preto, sua codificação é imediata e corresponde a letra P.

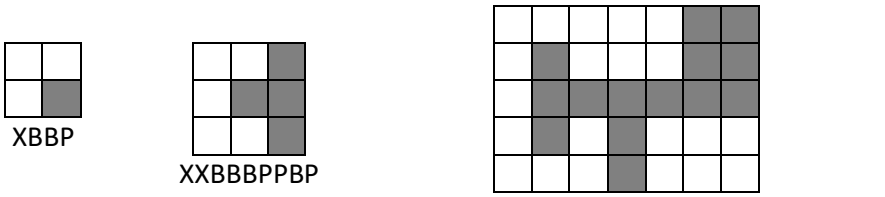
Com isso a última parte da imagem foi analisada, encerrando assim o processo de codificação da imagem dada como exemplo e gerando o código XXPBPBXPBBPXBPP.

Na Figura 7 vemos a imagem completa que foi utilizada como exemplo e como a sequência de letras da codificação foram concatenadas para constituir o código completo.

X XPBPB XPBBP XBP P
1º q. 2º q. 3º q. 4º q.

Figura 7 - A imagem usada como exemplo e o código correspondente.

A seguir (Figura 8) são apresentados alguns exemplos de imagens binárias com os seus respectivos códigos.



XBBP XXBBBPPBP XXXBBBPBXBPPXXBPBPPPPXXBPXBPBXBPB

Figura 8 - Outros exemplos de imagens e os códigos correspondentes.

Objetivos do trabalho

Desenvolver, utilizando linguagem C, um sistema para a codificação de imagens binárias que implemente a estratégia descrita no texto. A aplicação deve ser construída provendo uma interface do tipo CLI (*Command-line Interface*) permitindo ao usuário informar manualmente os dados da imagem ou o nome de um arquivo em formato PBM (*Portable Bitmap*), produzindo o código correspondente.

O formato *Portable Bitmap* (PBM)[1] foi adotado por conta de sua simplicidade e eficiência na forma de representar imagens binárias, pode ser representado tanto em binário quanto em modo texto[2] (ASCII).

No modo texto, o conteúdo de um arquivo PBM inicia com o "*magic number*" de valor é P1 (identificação que designa o formato do arquivo). Em seguida são apresentados dois valores numéricos, que correspondem a largura e altura da imagem. Os dados são apresentados a seguir, expressos por valores 0 (para branco) ou 1 (para preto), representando cada um dos pixels da imagem.

O conteúdo apresentado na Figura 9 corresponde um arquivo PBM de uma imagem com a letra "J" desenhada. Inclusive, esse texto pode ser copiado diretamente para o bloco de notas como uma imagem de entrada para a aplicação a ser desenvolvida.

Observe que nesse formato é possível encontrar linhas de comentário logo após o "*magic number*", sinalizadas pelo caractere de "#" e que devem ser descartadas.

[1] https://www.fileformat.info/format/pbm/egff.htm
[2] https://pt.wikipedia.org/wiki/Formato_Netpbm

```
P1
# Comentário: este é um exemplo de imagem da letra "J"
6 10
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
1 0 0 0 1 0
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

Figura 9 - Conteúdo de um arquivo em formato Portable bitmap (PBM) de uma imagem 6x10 contendo a letra "J".

Command-line Interface (CLI) e parâmetros do programa

Sendo uma aplicação do tipo CLI, as informações necessárias à execução do programa serão fornecidas através da linha de comando (como, por exemplo, o nome do arquivo de entrada que representa a imagem). Assim, é preciso que o seu programa seja capaz de ler e processar argumentos que sejam informados na linha de comando do sistema operacional.

Considerando que os parâmetros necessários ou mesmo a forma de utilização do programa podem ser desconhecidas ao usuário, caso nenhum argumento seja informado, o programa deverá apresentar na tela instruções de uso.

Na Figura 10 um exemplo de como um "help" da aplicação seria visualizado pelo usuário em um sistema Linux.

```
~/ap2/projeto$ ./ImageEncoder
Uso: ImageEncoder [-? | -m | -f ARQ]
Codifica imagens binárias dadas em arquivos PBM ou por dados informados
manualmente.
Argumentos:
-?, --help  : apresenta essa orientação na tela.
-m, --manual: ativa o modo de entrada manual, em que o usuário fornece
todos os dados da imagem informando-os através do teclado.
-f, --file: considera a imagem representada no arquivo PBM (Portable
bitmap).
```

Figura 10 - Tela de informações sobre como utilizar a aplicação na linha de comando.

A forma tradicional de uso é informar um arquivo em formato PBM. Assim, na linha de comando devem ser informados a opção `-f` seguido do nome do arquivo. Por exemplo:

```
~/ap2/projeto$ ./ImageEncoder -f imagem.pbm
```

Figura 11 - Linha de comando para que o programa leia o arquivo imagem.pbm e codifique-o.

Adicionalmente, caso seja informado um parâmetro `-m` ou `--manual` na linha de comando, o programa deverá considerar que a entrada de dados será feita de maneira manual pelo usuário e que nenhum arquivo de imagem será lido. Neste caso o programa começará a sua execução solicitando que o usuário informe as dimensões da imagem (largura e altura) e, em seguida, fará a leitura do valor correspondente a cada pixel da imagem.

Tanto no modo manual quanto aquele usando um arquivo PBM, o programa deverá apresentar na tela o código correspondente a imagem informada como entrada.

Observações importantes e considerações finais:

O programa deve ser implementado na linguagem C, ser modularizado e bem documentado. A entrega do trabalho deve ser feita pelo **GitHub Classroom**, observando-se a data limite para entrega.

O código entregue será avaliado de acordo com os seguintes critérios:

Rubrica	Não atendeu (não pontua)	Atendeu parcialmente (metade da nota)	Atendeu totalmente (nota completa)
Funcionamento do programa por meio de testes práticos (3,0 pontos)	Falhas de execução frequentes; funcionalidades críticas não implementadas; não há testes ou eles reprovam; casos de borda ignorados.	Funciona nos casos comuns, mas falha em casos específicos; erros intermitentes.	Passa em todos os testes previstos; comportamento consistente.
Fidelidade ao enunciado (1,0 ponto)	Solução diverge dos requisitos; altera escopo/fluxo; entradas/saídas diferentes das especificadas.	Implementa o essencial, mas omite regras/validações; suposições sem registrar; pequenas divergências em formato/fluxo.	Aderência integral aos requisitos; ambiguidades do enunciado tratadas e documentadas; entradas/saídas e fluxos exatamente conforme especificado.
Indentação, comentários e legibilidade (1,0 ponto)	Indentação inconsistente; linhas excessivamente longas; ausência de comentários úteis; difícil leitura.	Estilo majoritariamente correto, mas com trechos confusos; alguns comentários superficiais; padrões nem sempre seguidos.	Estilo consistente; comentários que explicam “por que” nas partes não óbvias; formatação clara;
Clareza na nomenclatura de variáveis (1,0 ponto)	Nomes genéricos/ambíguos (ex.: x, temp); abreviações obscuras; mistura de idiomas/convenções.	Maioria dos nomes é adequada, porém há casos pouco descritivos ou convenções inconsistentes.	Nomes autoexplicativos e consistentes; convenção definida (ex.: snake_case/camelCase); idioma padronizado.
Participação equitativa no GitHub (1,0 ponto)	Contribuição concentrada em 1 membro (>80%); histórico escasso; commits grandes e sem mensagem;	Todos contribuem, mas com desequilíbrio notável; mensagens de commit medianas.	Distribuição equilibrada de commits/PRs (≈40–60% entre principais); issues atribuídas; mensagens de commit claras; histórico transparente.

Rubrica	Não atendeu (não pontua)	Atendeu parcialmente (metade da nota)	Atendeu totalmente (nota completa)
Entrevista sobre o desenvolvimento (3,0 pontos)	Não comparece ou não demonstra autoria; desconhece arquitetura/decisões; respostas vagas.	Explica partes do trabalho, mas com lacunas; dificuldade em justificar decisões técnicas; demonstração parcial.	Domina arquitetura e decisões; demonstra o sistema com segurança; responde sobre testes, métricas e riscos de forma consistente.

Considere que as imagens de entrada terão como limite de tamanho máximo as dimensões 1024x768.

Este trabalho deve ser desenvolvido em trios ou duplas.