Documentação Trabalho Prático 0 da Disciplina de Estrutura de Dados

Daniel Oliveira Barbosa
Matrícula: 2020006450

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

danielolibar@gmail.com

1. Introdução

Esta documentação lida com o problema da conversão de uma imagem colorida (no formato .ppm (ASCII, P3)) para uma imagem em tons de cinza (no formato .pgm (ASCII, P2)). O objetivo do trabalho além de resolver o problema é recordar conceitos de programação em C/C++ e introduzir novos conceitos de abstração, desempenho e robustez. Para resolver o problema citado acima será necessário:

- Implementar uma solução para receber o nome do arquivo de entrada (no formato PPM) e o nome do arquivo de saída (no formato PGM) passados via terminal pela opção de linha de comando **-i**, e **-o** respectivamente.
- Criar uma estrutura de dados alocada de forma dinâmica para receber os dados lidos da imagem PPM.
- Criar uma solução para ler os dados do arquivo PPM de entrada e guardar na estrutura de dados mencionada no item anterior.
- Criar uma estrutura de dados alocada de forma dinâmica para receber os dados de uma imagem PGM após a conversão da imagem PPM para PGM.
- Criar uma solução para converter uma imagem PPM em uma imagem PGM através da função $Y = \frac{49}{255}(0.30R + 0.59G + 0.11B)$, e guardar na estrutura de dados mencionada no item anterior.
- Implementar uma função que gera um novo arquivo PGM e escreve os dados contidos na estrutura de dados criada para receber os dados de uma imagem PGM para esse novo arquivo PGM.

2. Implementação

Dado que imagens são representações de pontos de cor em um plano 2D, a escolha de usar uma matriz alocada dinamicamente para armazenar as imagens PPM e PGM veio naturalmente por sua natureza bidimensional e pela da variabilidade das dimensões da imagem.
Foram criadas classes que modelam as entidades do nosso problema, facilitando a abstração, modularização e organização do código. Desta maneira, apresentaremos nos subtítulos a seguir as classes utilizadas no projeto, juntamente com seus atributos e métodos.

OBS.: os atributos e métodos criados somente para fins de registro de acesso e análise de desempenho do programa serão omitidos por não influenciar na resolução do problema central de conversão de uma imagem PPM em uma imagem PGM.

Imagem	
Herda de nenhuma outra classe	
Atributos	**Métodos**
- **tipoImag** : string (indica o tipo da imagem P2 ou P3) - **numColunas** : int (indica o número de colunas da imagem) - **numLinhas** : int (indica o número de linhas da imagem) - **valorCorMax** : int (indica o valor máximo que uma cor da imagem pode atingir)	- **get_____()** (retorna o atributo) - **set_____()** (atribui o valor passado como parâmetro ao atributo)

PixelRGB	
Herda de nenhuma outra classe	
Atributos	**Métodos**
- **vermelho**: int (indica a intensidade de vermelho do Pixel) - **verde**: int (indica a intensidade de verde do Pixel) - **azul** : int (indica a intensidade de azul do Pixel)	- **get_____()** (retorna o atributo) - **set_____()** (atribui o valor passado como parâmetro ao atributo) - **converteRGBparaCinza()** (retorna o valor de cinza de um pixel convertido pela função Y = (49/255) * (0.30R + 0.59G + 0.11B)

ImagemPPM	
Herda da classe Imagem	
Atributos	**Métodos**

- **matrizPPM**: PixelRGB** (ponteiro para ponteiro que aponta para uma matriz de PixelRGB's, alocada dinamicamente)	- **ImagemPPM()** (construtor) - **~ImagemPPM()** (destrutor) - **getMatrizPPM()** (retorna ponteiro para ponteiro que aponta para a matriz PPM alocada dinamicamente) - **setElementoMatrizPPM()** (atribui os valores de vermelho, verde e azul passados como parâmetro para uma posição da matriz PPM) - **alocarMatrizPPMdinamicamente()** (faz a alocação dinâmica da matriz PPM) - **lerHeaderArquivoPPM()** (faz a leitura do tipo da imagem, número de linhas, número de colunas e valor de cor máxima do cabeçalho do arquivo PPM passado como parâmetro) - **lerImagemArquivoPPM()** (faz a leitura da imagem PPM contida no arquivo PPM passado como parâmetro) - **lerArquivoPPM()** (abre o arquivo PPM e chama os dois últimos métodos mencionados acima) - **converterParaPGM()** (faz a leitura da e conversão para cinza da matriz cinza e escreve o valor cinza na matriz PGM passada como parâmetro)

ImagemPGM	
Herda da classe Imagem	
Atributos	**Métodos**

- **matrizPGM**: int** (ponteiro para ponteiro que aponta para uma matriz de int's, alocada dinamicamente)	- **ImagemPGM()** (construtor) - **~ImagemPGM()** (destrutor) - **getMatrizPGM()** (retorna ponteiro para ponteiro que aponta para a matriz PGM alocada dinamicamente) - **setElementoMatrizPGM()** (atribui o valor de cinza passado como parâmetro para uma posição da matriz PGM) - **alocarMatrizPGMdinamicamente()** (faz a alocação dinâmica da matriz PGM) - **escreverHeaderArquivoPGM()** (faz a escrita do tipo da imagem, número de linhas, número de colunas e valor de cor máxima no cabeçalho do arquivo passado como parâmetro) - **lerImagemArquivoPGM()** (faz a escrita da imagem contida na matriz PGM em um arquivo passado como parâmetro) - **lerArquivoPPM()** (abre o arquivo PGM e chama os dois últimos métodos mencionados acima)

Devido a uma boa modularização, o programa principal conta com apenas 5 partes:

- A leitura das opções de comando do terminal,
- A declaração de objetos das classes Imagem PPM e PGM,
- A leitura do arquivo PPM através da chamada ao método **lerArquivoPPM(string nomeArquivoPPM)** da classe **ImagemPPM**,
    - Nesse processo, a matriz que guardará os dados da imagem PPM é alocada dinamicamente.
    - Depois é preenchida com os dados da imagem lidos do arquivo PPM.
- A conversão da imagem PPM para uma imagem PGM através da chamada ao método **converterParaPGM(ImagemPGM &imagemPGM)** da classe **ImagemPPM**, que chama o método **converteRGBparaCinza()** da classe **PixelRGB**.
    - Nesse processo, a matriz que guardará os dados da imagem PGM é alocada dinamicamente.
    - Depois é preenchida com os dados da matriz que guarda a imagem PPM após uma conversão para cinza
- A escrita do arquivo PGM através da chamada ao método **escreverArquivoPGM(string nomeArquivoPGM)** da classe **ImagemPGM**.
    - Nesse processo, os dados da imagem PGM que está contido na respectiva matriz é escrito em um arquivo PGM.

O programa foi desenvolvido utilizando a linguagem C++, compilador G++ da GNU Compiler

Collection e sistema operacional Ubuntu (versão 2204.1.6.0) através do Windows Subsystem for Linux 2 (a máquina utiliza como sistema operacional padrão o Windows 10 Pro 10.0.19043 Build 19043).

3. Análise de complexidade

As operações relevantes do programa são as leituras, escritas e conversões de RGB para cinza, e serão elas que usaremos para analisar a complexidade de tempo.

OBS-1: trataremos da complexidade de tempo na tabela abaixo apenas dos métodos que tiverem complexidade maior que O(1). Dessa forma, os métodos de get____(), set____(), lerHeaderArquivoPPM(), converteRGBparaCinza() e escreverHeaderArquivoPGM() não serão contemplados na tabela abaixo.
OBS-2: os métodos relacionados ao registro de acesso acessaMatriz___() também não serão contemplados na tabela abaixo.
OBS-3: os construtores, destrutores e métodos que fazem a alocação dinâmica alocarMatriz___dinamicamente() das matrizes também não serão contemplados na tabela abaixo por não terem operações relevantes ou terem complexidade de tempo O(1).

Já para a análise da complexidade de espaço, o item relevante para análise são as matrizes que armazenam as imagens PPM e PGM, e serão elas que usaremos para analisar essa complexidade.

Nome do Método	O que o Método Faz	Complexidade de Tempo	Complexidade de Espaço
lerImagemArquivoPPM() e lerArquivoPPM()	Faz a leitura do arquivo PPM e faz a escrita simultânea desses dados na matriz PPM.	O(**m*n**), sendo **m** e **n** o número de linhas e colunas da matriz PPM, respectivamente. Isso se dá pois é feito a escrita das cores vermelho, verde e azul de cada PixelRGB dentro de um loop aninhado em outro loop. Assim, essa escrita é feita passando por todos os elementos da matriz PPM de dimensões m,n., logo O(**m*n**).	O(**m*n**), sendo **m** e **n** o número de linhas e colunas da matriz PPM, respectivamente.

converterParaPGM()	Faz a leitura da matriz PPM, faz a conversão simultânea da escala RGB para cinza desses dados e faz a escrita simultânea desses dados na matriz PGM.	O(**m*n**), sendo **m** e **n** o número de linhas e colunas tanto da matriz PPM quanto da matriz PGM, respectivamente, pois ambas terão necessariamente as mesmas dimensões. Isso se dá pois é feito a leitura das cores vermelho, verde e azul de cada PixelRGB, a conversão para cinza e depois a escrita da cor cinza dentro de um loop aninhado em outro loop. Assim, essa leitura-->conversão-->escrita é feita passando por todos os elementos das matrizes PPM e PGM de dimensões m,n., logo O(**m*n**).	O(**m*n**), sendo **m** e **n** o número de linhas e colunas tanto da matriz PPM quanto da matriz PGM, respectivamente, pois ambas terão necessariamente as mesmas dimensões.
escreverArquivoPGM()	Faz a leitura da matriz PGM e faz a escrita simultânea desses dados no arquivo PGM.	O(**m*n**), sendo **m** e **n** o número de linhas e colunas da matriz PGM, respectivamente. Isso se dá pois é feito a leitura da cor cinza dentro de um loop aninhado em outro loop. Assim, essa leitura é feita passando por todos os elementos da matriz PGM de dimensões m,n., logo O(m*n).	O(**m*n**), sendo **m** e **n** o número de linhas e colunas da matriz PPM, respectivamente.

4. Estratégias de robustez

As estratégias de robustez criadas para evitar erros de input de usuários pelo terminal foram implementadas através de uma combinação de ”erroAsserts” e ”if statements”. Aqui estão as estratégias utilizadas para evitar esse tipo de erro:

- ErroAsserts utilizados para verificar que os nomes de arquivo de entrada e saída foram passados pelo terminal.
- ErroAssert e if statement utilizado para verificar que a flag -l requerendo os registros de acesso são passados em conjunto com a flag -p indicando o arquivo onde será feito o

registro desses acessos.
- If statement utilizado para evitar que o memlog seja ativado antes de ser iniciado, fazendo com que o arquivo de registro de acesso seja criado mas os registros de acesso em si não sejam escritos no arquivo.

As estratégias de robustez criadas para evitar erros do arquivo PPM foram implementadas através de ”erroAsserts”. Aqui estão as estratégias utilizadas para evitar esse tipo de erro:

- ErroAssert utilizado para verificar que o arquivo PPM foi aberto corretamente.
- ErroAsserts utilizados para verificar que o formato do arquivo PPM era correto (formato correto: P3), se as dimensões da imagem PPM eram não nulas e se o valor de cor máxima estava correta (valor correto: 255).
- ErroAsserts utilizados para verificar que qualquer cor da imagem contida no arquivo PPM esteja no intervalo de [0, 255].

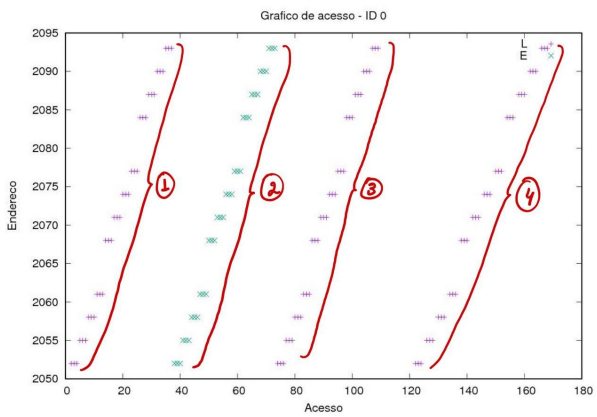
A estratégia de robustez criada para evitar erros do arquivo PGM foi implementada através de um ”erroAssert”. Essa estratégia foi apenas verificar que o arquivo PGM tivesse sido aberto corretamente.

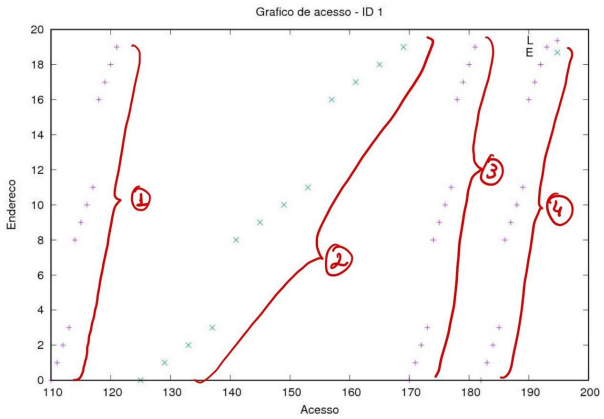
5. Análise Experimental

Para a análise dos acessos às matrizes foi usado o memlog, com auxílio do analisamem e o gnuplot para a geração dos gráficos. Usaremos um exemplo de imagem com 3 linhas e 4 colunas para podermos fazer a análise. Outro item importante para entender a análise são as diferentes fases da análise:

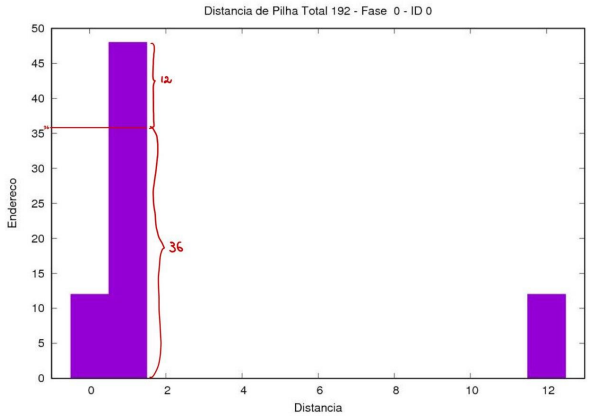
- A fase 0 compreende:
  - Leitura do arquivo PPM
  - Escrita simultânea dos dados lidos do arquivo PPM na matriz PPM
- A fase 1 compreende:
  - Leitura da matriz PPM
  - Conversão simultânea dos dados lidos da matriz PPM para cinza
  - Escrita simultânea do cinza convertido para a matriz PGM
- A fase 2 compreende:
  - Leitura da matriz PGM
  - Escrita simultânea dos dados lidos da matriz PGM no arquivo PGM

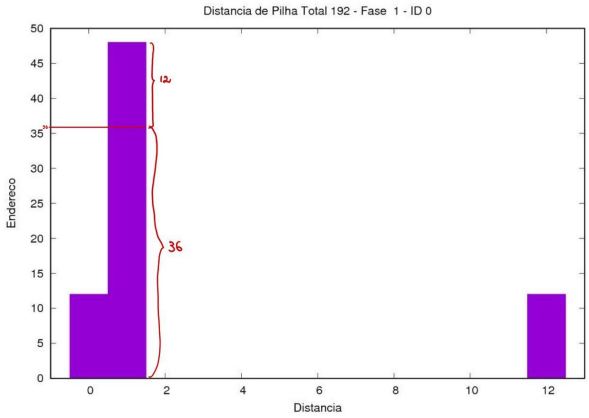
Veja as imagens a seguir (caso não seja possível dar zoom, clique na palavra link para visualizar essas imagens):





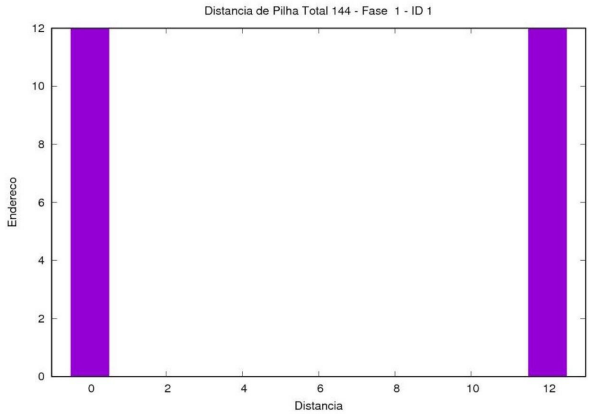
Esses são os gráficos de acesso da matriz PPM (ID = 0) e da matriz PGM (ID = 1). Em ambos os gráficos, os conjuntos de acessos indicados por 1 e 3 são acessos gerados métodos acessaMatriz___() que tem objetivo único auxiliar na geração dos gráficos de distância de pilha. Assim, devem ser desconsiderados. Já os conjuntos de acessos indicados por 2 e 4 indicam a escrita nas matrizes e a leitura das matrizes, respectivamente.
Um detalhe importante é o fato de que no gráfico de acesso da matriz PPM (ID = 0), vemos que há 3 acessos ao mesmo endereço a cada elemento da matriz, indicados por “xxx” ou por “+++”. Isso sinaliza que as 3 cores do PixelRGB estão sendo escritas ou lidas. Em contraste, no gráfico de acesso da matriz PGM (ID = 1), há apenas 1 acesso ao mesmo endereço a cada elemento da matriz, indicado por “x” ou por “+”. Isso sinaliza que apenas a cor cinza está sendo escrita ou lida.
Agora veremos os gráficos de distância de pilha da matriz PPM (ID = 0) nas fases 0 e 1. (caso não seja possível dar zoom, clique na palavra link para visualizar essas imagens)

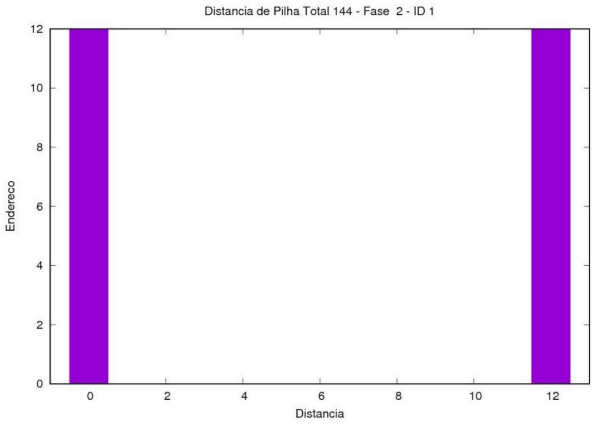




Veja que os gráficos são idênticos a não ser pelo fato de que os acessos da fase 0 são para escrita na matriz e na fase 1 são para leitura da matriz. O grupo de acessos com distância de pilha 0 é referente ao método acessaMatrizPPM(), que acessa os 12 elementos da matriz, porém devem ser desconsiderados. Já o grupo de acessos com distância de pilha 12 é referente ao acesso de cada uma das 12 posições da matriz que sempre estão no fundo da pilha e portanto a uma distância de pilha de 12. Por último, o grupo de acessos com distância de pilha 1 é referente ao acesso das 3 cores RGB de cada PixelRGB das 12 posições (36 acessos) da matriz

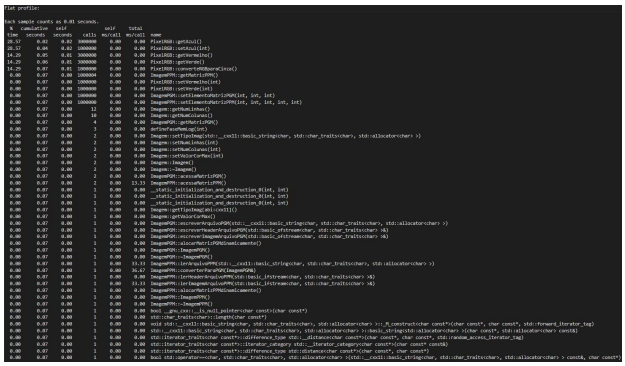
mais 12 acessos por posição da matriz, resultando em 48 acessos de distância de pilha 1. Agora veremos os gráficos de distância de pilha da matriz PGM (ID = 1) nas fases 1 e 2. (caso não seja possível dar zoom, clique na palavra link para visualizar essas imagens)

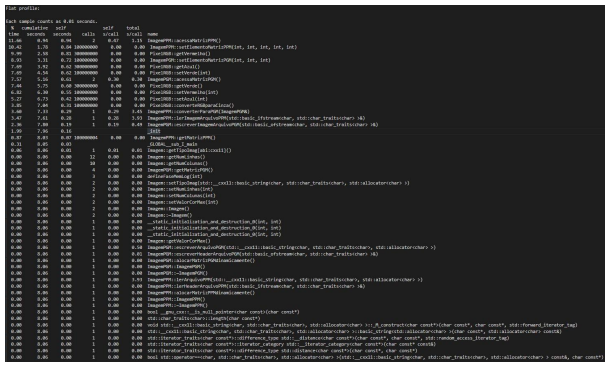




Veja que os gráficos são idênticos a não ser pelo fato de que os acessos da fase 1 são para escrita na matriz e na fase 2 são para leitura da matriz. O grupo de acessos com distância de pilha 0 é referente ao método acessaMatrizPGM(), que acessa os 12 elementos da matriz, porém devem ser desconsiderados. Já o grupo de acessos com distância de pilha 12 é referente ao acesso de cada uma das 12 posições da matriz que sempre estão no fundo da pilha e portanto a uma distância de pilha de 12.

Usando o gprof, fizemos um primeiro teste de passando uma imagem com dimensões 1000x1000 e depois fizemos um outro teste com uma imagem de dimensões 10000x10000. Veja as imagens do “flat profile” do gprof abaixo. (caso não seja possível dar zoom, clique na palavra link para visualizar essas imagens)





Veja que, no primeiro teste, o programa demorou aproximadamente 0.07 segundos para ser executado. Já no segundo teste, o programa demorou aproximadamente 8.06 segundos para ser executado. Devido a precisão do gprof ser até 0.01 segundos, podemos considerar que houve um aumento de 100 vezes no tempo de execução de um teste para outro, apesar de ter aumentado apenas em 10 vezes o número de linhas e colunas. Assim, foi possível atestar empiricamente que a complexidade de tempo do programa é de fato de $O(n^2)$.

6. Conclusão

O problema a ser solucionado neste trabalho prático era de fazer a conversão de uma imagem no formato PPM (colorida) para uma imagem no formato PGM (cinza). O maior desafio do trabalho foi conseguir abstrair as características e ações relevantes de imagens PPM e PGM. Dessa maneira, atingimos o objetivo de abstrair as imagens em seus atributos e métodos e também implementar uma estrutura de dados alocada dinamicamente que fosse ideal para a resolução do problema.
Com esse trabalho, pude revisar vários conceitos de programação orientada a objetos como modularização, herança, encapsulamento, além de conceitos básicos de programação em C++. Também foi possível aproveitar para praticar o uso de boas práticas e do debugador, itens que facilitam tanto a escrita quanto a revisão do programa. Em suma, o trabalho permitiu a visualização do início ao fim de um projeto aplicando a estrutura de dados necessária para resolvê-la.

7. Bibliografia

- Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 3ª Edição, Cengage Learning, 2011
- Cormen , T., Leiserson, C, Rivest R., Stein, C. Introduction to Algorithms, Third Edition, MIT Press, 2009. Versão Traduzida:Algoritmos – Teoria e Prática 3a. Edição, Elsevier, 2012
- https://www.youtube.com/watch?v=DUZHe1nz6bs&ab_channel=CSExplained
- Vídeo Aulas e Slides disponibilizadas na Metaturma da disciplina de Estrutura de Dados

8. Instruções de compilação e execução

- Vá para o diretório raiz do projeto (/TP)
- Digite na linha de comando (para limpar arquivos desnecessários do diretório): **make clean**
- Digite na linha de comando (para compilar o programa): **make**
- Temos algumas opções para executar o programa
    - Para executar o programa sem criar o arquivo de registros de acesso, digite na linha de comando: **./bin/programa -i <nome do arquivo ppm de entrada> -o <nome do arquivo pgm de saída>**
    - Para executar o programa criando o arquivo de registros de acesso, porém sem os registros, digite na linha de comando: **./bin/programa -i <nome do arquivo ppm de entrada> -o <nome do arquivo pgm de saída> -p log.out**
    - Para executar o programa criando o arquivo de registros de acesso com os registros, digite na linha de comando: **make**