Trabalho Prático 0

Conversor de imagens PPM para PGM

Matheus Grandinetti Barbosa Lima - 2021067496

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brazil

matheusgrandinetti@gmail.com

1. Introdução

Esta documentação lida com o problema da conversão de uma imagem colorida do tipo ppm para uma imagem preto e branco, do tipo PGM. Para concretizar a funcionalidade proposta, o programa foi dividido em três principais partes. A primeira, o *loading* da imagem, tem como objetivo recolher as informações da imagem a ser convertida. A segunda converte os valores de pixel colorido para a escala de cinza. A terceira gera um novo arquivo do tipo PGM, esse arquivo é a imagem final, tratada e produzida pelo programa.

2. Método

2.1. Configurações da máquina

Sistema operacional: WSL - Ubuntu 20.04 LTS
Linguagem de programação: C++
Compilador: G++ / GNU Compiler Collection.
Processador: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
Memória RAM: 16 GB

2.2. Estruturas de dados

Na implementação desse trabalho, a estrutura de dados utilizada foi a matriz. O uso dessa estrutura específica teve em vista uma abstração mais concisa e palpável de um tipo abstrato imagem, com o qual foram realizadas as operações de *loading* e conversão.

A matriz foi alocada dinamicamente no programa, sendo referenciada dentro das classes implementadas por meio de um ponteiro para ponteiros. É criada após a leitura do header contido no arquivo ppm, pois dessa forma sabemos melhor as informações de dimensões e tamanho dessa matriz. Dentro dela, apelidada de *pixelMatrix* no programa, armazenamos pixels, outro TAD criado para facilitar a abstração de uma imagem.

2.3. Classes

Foram utilizadas duas principais classes na implementação desse trabalho, a classe Pixel e a classe Image.

A classe Pixel é utilizada para armazenar os valores de cor de cada pixel da imagem, possuindo funções básicas de *set* e *get* de informações. Por meio dessa classe, é possível criar uma forma mais organizada de armazenamento de conteúdo da imagem escolhida.

A classe imagem é a principal do programa, utilizada para armazenar pixels e aplicar as operações sobre a imagem desejada. É aqui onde estão guardadas as informações colhidas do header do arquivo PPM e a matriz de pixels, onde é possível realizar as conversões necessárias.

2.4. Funções

O código possui 3 principais funções, a *loadImage()*, *convertImage()*, *generatePgmImage()*.

A função *loadImage* é utilizada na leitura de um arquivo PPM, sendo a responsável por recolher os valores do *header* do arquivo, contendo tipo, dimensões e valores máximos de cada imagem, e as informações sobre os pixels RGB. Os valores são lidos por meio de *file streams* da linguagem c++ e armazenados na matriz *pixelMatrix*, podendo ser acessados posteriormente para análise.

A função *convertImage* tem como objetivo a conversão dos valores RGB da matriz de pixel para valores na escala cinza. Dessa forma, analisamos cada pixel individualmente, realizando operações por toda a nossa imagem.

A função *generatePgmImage* é utilizada na criação de um arquivo PGM, utilizando *file streams* para a escrita das informações de imagem oriundas da aplicação da função *convertImage*. É realizada uma varredura na *pixelMatrix*, recolhendo os valores na escala de cinza de cada pixel da imagem, sendo transcritos para o novo arquivo PGM.

Outras funções como *getters* e *setters* são utilizadas como forma de facilitar a manipulação de informações pertencentes a estruturas privadas.

3. Análise de complexidade

loadImage() - Complexidade de tempo: Dentro dessa função realizamos varreduras simples no header do arquivo, possuindo complexidade proporcional a O(1). Para a leitura de cada pixel da imagem foi utilizada a chamada de dois loops aninhados com especificações de parada relativas ao tamanho de cada imagem de entrada. Dessa forma, a complexidade assintótica da função é O(n^2).

loadImage() - Complexidade de espaço: A análise de espaço dessa função é proporcional às dimensões da imagem que, por serem mutáveis, podem ser descritas por O(n^2).

convertImage() - Complexidade de tempo: Essa função é composta por dois loops aninhados que varrem a matriz de pixels, aplicando a fórmula de conversão individualmente. A fórmula de conversão por si só possui custo assintótico proporcional a O(1), pois independe do tamanho de qualquer entrada, porém a aplicação dessa, pixel a pixel dentro dos loops citados, gera um custo assintótico total de O(n^2).

convertImage() - Complexidade de espaço: A análise de espaço dessa função é proporcional às dimensões da imagem que, por serem mutáveis, podem ser descritas por O(n^2).

generatePgmImage() - Complexidade de tempo: A função *generatePgmImage* tambem possui dois loops aninhados para realizar a varredura na matriz de pixels, possuindo nessa parte um custo assintótico de O(n^2). Porém, nessa função é realizada a chamada da *convertImage* que também possui custo O(n^2), fator que não altera o custo assintótico final, sendo também O(n^2).

generatePgmImage() - Complexidade de espaço: A análise de espaço dessa função é proporcional às dimensões da imagem que, por serem mutáveis, podem ser descritas por O(n^2).

4. Análise de robustez

Como forma de padronizar e deixar o código mais legível, foram adotadas algumas medidas de indentação e nomeação de variáveis padrão. As variáveis estão nomeadas com base no formato camelCase, todas em inglês. Além disso, foi utilizado o formatador Clang-Format, para que dessa forma, o código tivesse aparência mais uniforme e estável.

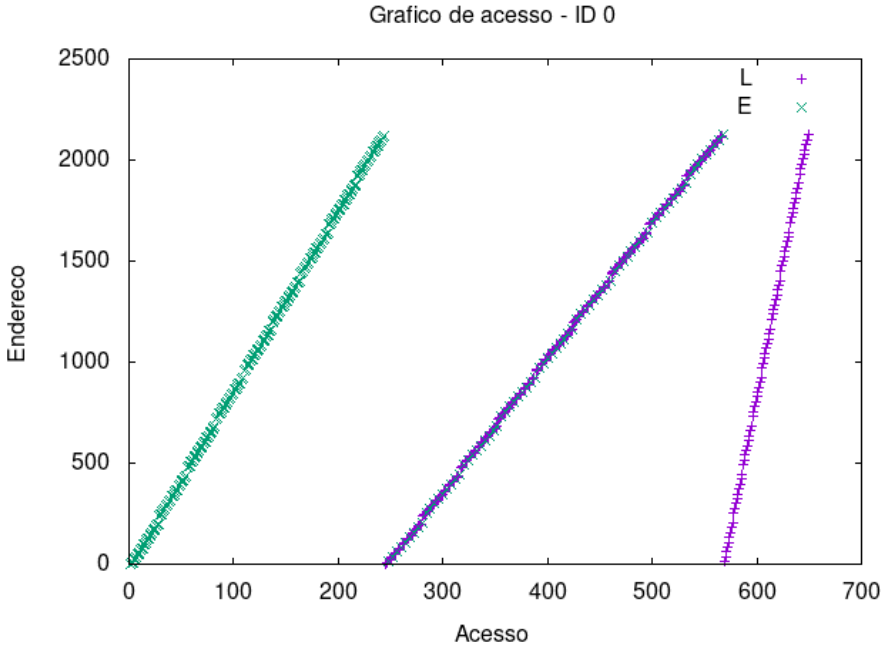
Analisando a parte do tratamento de erros, foi utilizada a biblioteca msgassert.h, disponibilizada pelos professores da disciplina, sendo aplicada em dois principais cenários. No âmbito dos arquivos, um *erro assert* foi aplicado na verificação da abertura da imagem, impedindo a continuidade do programa em situações de erro. No âmbito das imagens, outro assert foi utilizado com o objetivo de impedir a análise de figuras com dimensões nulas. Na função *getPixelValue* foi utilizado um assert para verificar o return de valores inválidos de cores, finalizando o programa caso não esteja entre as cores válidas.

5. Análise experimental

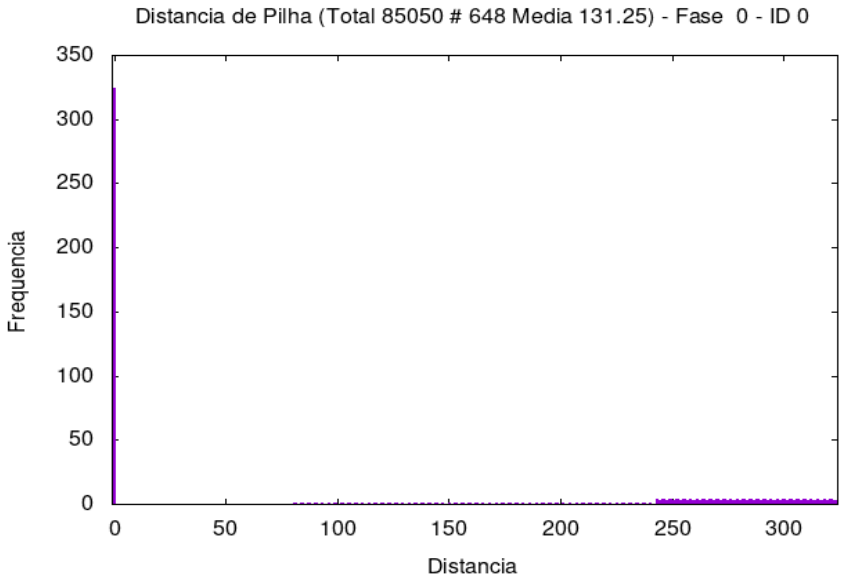
Antes da aplicação do programa

Depois da aplicação do programa

Utilizando as ferramentas memlog, analisamem, e gnuplot, obtemos a relação de uso de memória na conversão de uma imagem teste.ppm, 9x9.



De acordo com o gráfico apresentado, é possível notar com exatidão as três funções implementadas no programa, bem como a forma como se comportam. Os pontos sinalizados pela cor verde estão relacionados com a criação da matriz de pixels, sendo caracterizada pela escrita de memória. A reta do meio possui a cor roxo escuro, e corresponde a função de conversão dos pixels que, no código em destaque, acessa as cores RGB e armazena o valor dessa operação no valor de cinza. A cor apresentada é na verdade a sobreposição dos pontos de escrita e leitura que acontecem alternadamente na função, fazendo com que pareçam ser da mesma cor. A terceira reta é relativa à função de escrita da imagem PGM, fazendo com que o gráfico de acesso seja composto por marcações de leitura, ligados aos valores de cinza da matriz de pixel.



6. Conclusão

Dessa forma, o programa criado de acordo com as especificações propostas resolve de forma sucinta e objetiva o problema da conversão de imagens PPM para PGM, aplicando os conceitos vistos em sala de aula e utilizando tecnologias referentes ao memlog, analisamem e gnuplot.

Durante sua modelagem e execução, as maiores dificuldades encontradas foram da revisão de conceitos relativos ao acesso e leitura de arquivos em c++ e o uso das bibliotecas fornecidas pelos professores, a memlog e a analisamem.

7. Bibliografia

Gisele L Pappa e Wagner Meira Jr. (2020). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

8. Instruções de compilação e execução

Diretamente do diretório *root* utilize o comando '**~$ make**' para realizar a compilação do programa. Caso deseje limpar o diretório /obj/ que contém os arquivos .o gerados na compilação, digite '**~$ make clean**' no terminal.

Para rodar o arquivo executável, acesse o diretório /bin/ que contém o target "program". A execução do programa deve conter a declaração de algumas tags para o seu funcionamento, sendo elas: -i "imagem de entrada" -o "imagem de saída" -p "log.out" -l. a tag -i é utilizada para acessar a imagem PPM que será convertida, já a tag -o é utilizada nada declaração do nome do arquivo de saída, no nosso caso uma imagem PGM. As tags -p -l estão ligadas ao funcionamento e uso da biblioteca memlog, sendo utilizadas para iniciar (-p) e ativar (-l) a mesma.

Um exemplo de linha de comando para uso do executável seria:
~$ ./program -i imagem.ppm -o output.pgm -p log.out -l