Trabalho Prático 0

Introdução

O problema contido no trabalho prático 0 era a conversão de uma imagem colorida, no formato .PPM, para uma imagem em escala de cinza, com 50 tons diferentes, no formato .PGM.

Com esse objetivo foi feito um plano de desenvolvimento que envolvia 3 tipos abstratos de dados, cada um com seus métodos (o que ele faz) e seus atributos (as informações que ele guarda) próprios, visando auxiliar e possibilitar a resolução do problema.

Método

Main.cc:

O arquivo main.cc é o responsável por iniciar todas as operações de conversão dos arquivos, mas não é ele que, de fato, lê, converte ou escreve a imagem.

Na hora de iniciar o programa, são passados alguns argumentos, sendo responsabilidade da função main registrá-los e executá-los. Além disso, a função main fica encarregada de iniciar o memLog, para a análise em tempo de execução do programa, abrir o arquivo da imagem a ser lida, criar o arquivo da imagem já convertida e, chamar o método ConvertImage que realmente fica encarregado da conversão da imagem.

Estruturas de dados:

Como mencionado anteriormente, foi escolhida a divisão da resolução em 3 tipos de abstratos distintos, as classes Pixel, Image e Converter.

1 - Pixel

O Pixel é a estrutura de dados mais simples criada, com poucos dados a serem guardados e poucas ações a serem realizadas por ele.

Cada Pixel possui 3 atributos, todos variáveis de números inteiros, sendo eles vermelho (red), verde (green) e azul (blue). Todos os atributos são privados, o que significa que não é possível acessá-los ou modificá-los fora da estrutura Pixel.

Como métodos, Pixel tem o construtor, getters e setters. O construtor age na hora de criação do objeto, com instruções para sua inicialização. No caso do Pixel, não há nenhuma especificação, só criando o objeto. Os getters e setters são necessários devido ao fato de os atributos serem privados. Por não ser possível alterar os atributos fora da estrutura Pixel, são criados

métodos auxiliares, contidos na estrutura, que recebem como parâmetro valores para os quais alterar os atributos do Pixel (setters), ou que retornam os valores atuais dos atributos (getters).

Nesse caso, existe um get para cada atributo (GetR, GetG e GetB), mas somente um set, o SetRGB, sobrecarregado (métodos com funcionalidades e/ou atributos diferentes, que possuem o mesmo nome), para alterar todos os atributos no mesmo método de duas formas possíveis: recebendo 3 parâmetros diferentes e alterando cada atributo para um parâmetro, utilizado para dar cor ao pixel, ou recebendo somente 1 parâmetro e alterando todos os atributos para esse, utilizado para atribuir um tom de cinza ao pixel.

2 - Image (Imagem)

A estrutura imagem, já é mais complexa, contando com mais, e mais diferenciados, atributos, assim como alguns métodos diferentes dos da estrutura Pixel.

Assim como no tipo abstrato de dado anterior, a Imagem possui somente atributos privados, sendo eles height (altura da imagem), width (largura da imagem), ambas variáveis de tipo inteiro, além de um atributo matrix, que é uma matriz de variáveis do tipo Pixel, de tamanho altura x largura.

Os métodos da Imagem são, em sua maioria, simples, como os métodos do Pixel. Existe um construtor, que agora inicializa os valores da altura e largura, além de criar a matriz de pixels por meio de alocação dinâmica (gerenciamento manual da alocação de memória para as variáveis), baseada na altura e largura e também um destrutor, o qual realiza a operação contrária de um construtor, realizando operações na hora de destruição do objeto, ao invés de criação. O destrutor de Imagem se encarrega de desalocar os pixels contidos na matriz, sendo necessário devido à alocação dinâmica.

Além desses, existem também os getters e os setters para Imagem. O SetPixel é sobrecarregado e tem a função de chamar o SetRGB de um pixel específico da matriz, informando qual linha e coluna o pixel está na matriz, assim como a cor do pixel, que pode ser com mais 3 parâmetros, no caso de querer o Pixel colorido, ou 1, em escala de cinza. Nos getters, estão o GetHeight, para pegar a altura da matriz, o GetWidth, para pegar a largura, o GetPixel, para pegar o Pixel da matriz, com linha e coluna informados por parâmetro e, por último, o GetPixelAddress, que retorna o endereço no qual o Pixel informado por linha e coluna está alocado na memória.

3 - Converter (Conversor)

A classe mais abrangente, o Conversor, possui métodos mais complexos que os vistos até então, mas somente dois atributos, também privados.

Os atributos são do tipo Imagem, sendo eles “toConvert”, que guarda os dados da imagem original, ainda a ser convertida para escala de cinza, e “converted”, que guarda os dados da imagem já convertida.

Dentre os métodos, temos:

- ReadImage - Lê a imagem recebida como parâmetro, salvando os dados da imagem original no atributo de tipo Imagem “toConvert”.
- GrayScale - Transforma a imagem original, contida no atributo de tipo Imagem “toConvert”, em escala de cinza e armazena os dados no atributo de tipo Imagem “converted”.
- WriteImage - Escreve no arquivo passado por parâmetro a imagem convertida, contida no atributo de tipo Imagem “converted”.
- CheckRange - Chamado nos métodos ReadImage e GrayScale, tem a função de verificar se os valores para os atributos dos pixels estão dentro dos valores permitidos.
- ConvertImage - Esse método é responsável por gerenciar todo o processo de conversão, chamando do ReadImage até o WriteImage, sendo chamado pela função main, com os arquivos de entrada e de saída como parâmetros.

Análise de Complexidade

Complexidade de Espaço:

A complexidade de espaço está associada à quantidade de memória utilizada pelas variáveis do programa. O objeto abrangente, o Conversor, possui como atributos, 2 outros objetos do tipo Imagem, os quais ocupam a mesma quantidade de memória, por terem necessariamente as mesmas alturas e larguras. O objeto Imagem, por sua vez, ocupa uma quantidade de espaço diretamente proporcional à altura e largura da matriz de pixels contida nele. Com isso, conclui-se que a complexidade de espaço sempre será limitada superior e inferiormente pelo produto da altura (h) e largura (w) da matriz de pixels contida em cada atributo Imagem (nunca usará mais ou menos espaço que o produto da altura e largura da matriz de pixels, multiplicado por uma constante), ou seja, a complexidade de espaço é $\Theta(h*w)$ ou, se h e w forem iguais, $\Theta(n^2)$.

Complexidade de Tempo:

Para a complexidade de tempo, definiremos as operações relevantes (operações mais realizadas em um algoritmo) como as de ler um Pixel ou atribuir valores ao Pixel, as quais são realizadas durante os métodos ReadImage, GrayScale e WriteImage.

Em cada um desses métodos, existem 2 loops feitos por meio do for. O loop externo começa em 0 e vai até h (altura) - 1, sendo executado h vezes. O loop interno, por sua vez, começa em 0 e vai até w (largura) - 1,

para um total de w execuções, mas como ele se encontra dentro do primeiro loop, cada execução do primeiro loop, o segundo é executado w vezes, ou seja, o código dentro do segundo loop é executado um total de h*w vezes. Dentro desses 2 loops se encontram as operações de ler um Pixel ou atribuir valores ao Pixel.

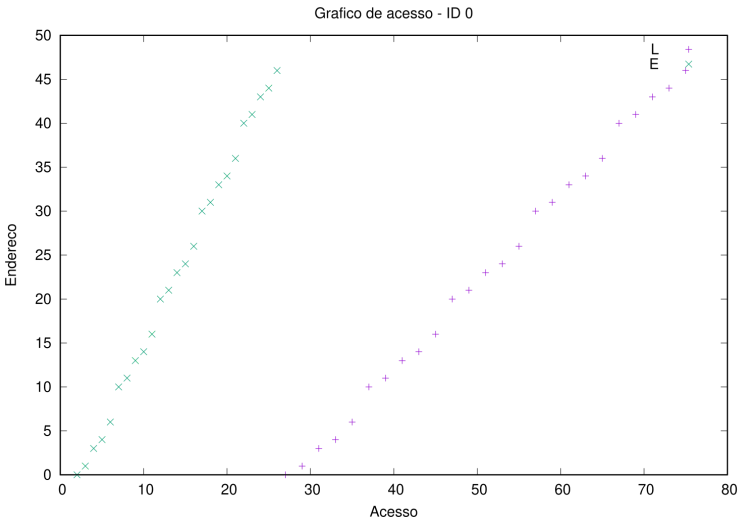
No método ReadImage, são atribuídos valores a um Pixel a cada execução do loop interno, portanto, no método inteiro, isso é feito h*w vezes. No método GrayScale, um Pixel é lido e outro tem valores atribuídos a cada loop, para mais um total de h*w vezes por cada uma dessas operações. Para finalizar, no método WriteImage, um Pixel é lido a cada loop, para mais um total de h*w vezes. Assim, apesar de não contar todas as operações feitas dentro dos loops, é possível ver que são sempre realizadas, dentro deles, um número de operações diretamente proporcional ao produto da altura e largura da imagem lida/escrita, por isso, a complexidade de tempo é, assim como a de espaço, limitada superior e inferiormente pelo produto da altura e da largura, sendo então, $\Theta(h*w)$ ou, se h e w forem iguais, $\Theta(n^2)$.

Estratégias de Robustez

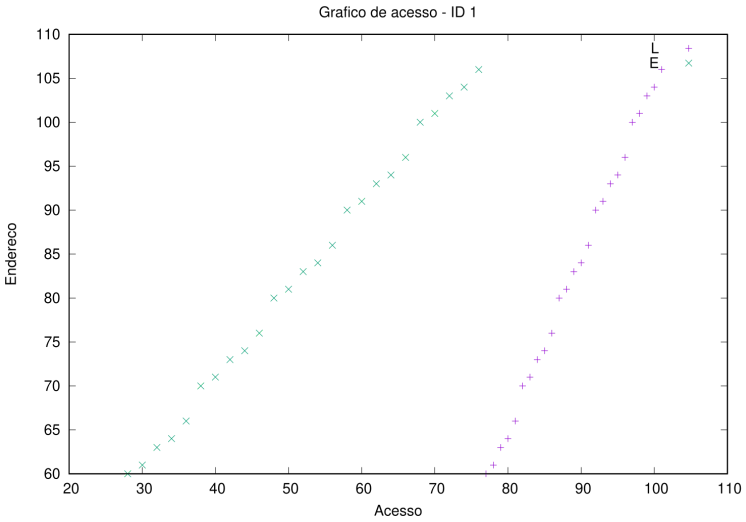
Para evitar a interrupção desnecessária do programa, foi adicionada uma função para checar se os argumentos contendo o nome de entrada e saída do programa contém a extensão do arquivo requerida pelo programa, não interrompendo a execução se esse não for o caso.

Análise Experimental

Foi analisado o desempenho do programa com a biblioteca memLog, gerando gráficos por meio do analisaMem.



Como é possível perceber, o acesso à memória se dá de forma sequencial, por estar acessando cada Pixel da matriz depois do Pixel em posição anterior, por meio do loop do for.



Isso ocorre nas duas matrizes, mas as sequências se diferenciam na inclinação, uma vez que existem diferentes quantidades de operações em cada loop.

Conclusões

No trabalho, foi criado um programa para a conversão de uma imagem .PPM em uma imagem .PGM. Durante todo esse processo, houve um melhor aprendizado sobre formatos de arquivos e formas de manipulá-los por meio de código. Além disso, o trabalho possibilitou um aprendizado da biblioteca memlog e da utilização do analisamem, aumentando o conhecimento sobre os conceitos de localidade de referência, distância de pilha e complexidade de tempo e espaço.

Bibliografia

Para a função endWith, foi utilizado como base o link: https://stackoverflow.com/questions/874134/find-out-if-string-ends-with-another-string-in-c

Para funções padrão de c++, foi utilizado como base o link: https://cplusplus.com/forum/

Para detalhes sobre o analisamem, foi utilizado o link: https://docs.google.com/document/d/1h-yeWDz2FjBVvMgI1gfXyV_ogOmwlpneKEfq4yHihoA/edit

Instruções para Compilação e Execução

Para a compilação do programa em sistema Linux, primeiro é necessário abrir uma janela do terminal dentro da pasta TP0. Após feito isso, basta digitar a linha de comando “make” e o programa será compilado.

Para sua execução em ambiente Linux, é necessário também abrir a janela do terminal na pasta TP0, para então digitar a linha de comando “./bin/run.out ARGUMENTOS”, onde argumentos são os argumentos a seguir, somente o último sendo opcional:

- “-i entrada” ou “-i entrada.ppm” - O argumento “-i” é seguido do nome do arquivo da imagem que se deseja converter, que deve estar no diretório TP0, com ou sem o .ppm, que indica a extensão.
- “-o saída" ou “-o saída.pgm" - O argumento “-o” é seguido do nome do arquivo da imagem que se deseja converter, com ou sem o .pgm, que indica a extensão.
- “-p registro” ou “-p registro.out” - O argumento “-p” é seguido do nome do arquivo em que se deseja fazer o registro de desempenho, com ou sem o .out, que indica a extensão
- “-l” - O argumento “-l” deve ser adicionado somente se o usuário desejar que seja registrado os acessos à memória durante a execução do programa, caso contrário, será registrado apenas o tempo de execução do programa.

Dessa forma, o programa será executado, e a imagem convertida, criando o novo arquivo de imagem, com o nome informado e a extensão .PGM, e o registro de desempenho no diretório TP0.