Trabalho Prático 0
Estrutura de dados

Vinicius Leite Censi Faria

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

vlcfaria@ufmg.br

1. Introdução

O trabalho proposto, com objetivo de recordar fundamentos de desenvolvimento em C/C++ e introduzir conceitos como TADs (Tipos Abstratos de Dados), se trata da conversão de imagens no formato de PPM, com cores, para o formato PGM, em um certo espectro de preto e branco. Além disso, também se estabelece como objetivo analisar a complexidade do programa, juntamente com sua comprovação na parte experimental do trabalho. Esses formatos de arquivos são caracterizados por uma sequência de números representando, para cada pixel, os tons de vermelho, azul e verde no caso do PPM, e cinza no outro formato.

Como solução, optou-se pela criação de TADs para os dois formatos de arquivos, cada um com uma matriz com valores representando os pixels na tela, além de outros atributos que são essenciais para a construção da imagem. A partir dessas instâncias abstratas, foram desenvolvidas funções separadas e consolidadas para fazer a leitura, conversão e escrita dos arquivos. Além disso, foram implementados diversos asserts para garantir a robustez, além de registros de memória e desempenho para o programa.

2. Método

2.1 Estruturas de dados

Como uma prática introdutória, a única “estrutura de dados” utilizada na prática foi uma simples matriz alocada dinamicamente para aceitar valores durante o *runtime*. Para obter maior abstração, implementou-se da maneira mais convencional, alocando diversos vetores, cujos elementos representam as colunas de uma linha, dentro de outro vetor, cujos elementos representam as linhas. O acesso de um elemento a_{ij} se dá pelo acesso ao elemento “matriz[i][j]”.

2.2. Classes

Em primeiro lugar, como a classe mais básica, foi implementado a classe PixelRGB, abstração básica de um pixel da imagem no formato PPM. Possui atributos vermelho, verde e azul, além do construtor padrão, construtor personalizado, e o método que retorna seu valor no espectro de cinza desejado, seguindo a fórmula:

$$Y = \frac{49}{255}(0.30R + 0.59G + 0.11B)$$

A partir dessa abstração, foi criado um TAD PGM, que possui uma matriz, alocada dinamicamente, de PixelRGBs, além de valores inteiros de tamanho, largura e o valor máximo, que caracterizam as propriedades da imagem e são especificados logo nas

primeiras linhas do arquivo e inicializados logo no construtor. Além disso, para adaptar o TAD para o Memlog, foi introduzido um atributo inteiro de id.

Assim como no TAD anterior, o TAD PPM é composto por uma matriz alocada dinamicamente, dessa vez de inteiros, representando a tonalidade de cinza de cada pixel na imagem dentro de um espectro definido. De maneira similar, também há inteiros representando o tamanho, largura, valor máximo e um id, que são inicializados diretamente no construtor.

2.3. Procedimentos

O programa contém 3 funções para criação, manipulação e escrita dos formatos solicitados.

O primeiro procedimento, readFromFile tem como parâmetro uma string representando o nome do arquivo de entrada e um inteiro representando o id do objeto ppm criado. A função inicialmente lê os dados iniciais do arquivo, passando para o construtor do PPM. Após isso, inicia a leitura de pixels e insere os valores de vermelho, verde e azul na medida que vai percorrendo a matriz do TAD.

No caso do segundo procedimento, convertPpmToPgm, toma como parâmetro um ponteiro para uma instância PPM e um id, que irá representar o id único do novo PGM formado. A função inicializa dinamicamente um nova instância PGM com a mesma largura e altura do PPM informado, e depois vai percorrendo a matriz de PixelRGBs invocando sua função convertToGray e atribuindo-o para a matriz de inteiros do PGM.

No terceiro método, que toma como parâmetro o nome do arquivo de saída e um ponteiro para PGM, escreve uma instância do TAD para um arquivo, permitindo sua visualização. Após abrir o arquivo, ele escreve os metadados que possui e percorre sua matriz, inserindo no arquivos os inteiros que representam os tons de cinza.

3. Análise de Complexidade

Considerando as funções relevantes e solicitadas pelo projeto, temos:

Função Ppm::Ppm (Construtor do TAD Pgm): A função realiza operações constantes O(1), porém apresenta um laço para a alocação das colunas de cada linha. Sendo assim, a complexidade de tempo é formada por: O(1) + O(n) = O(Max(n,1)) = O(n), o que resulta em Θ(n), sendo n a altura da imagem. Na complexidade de espaço, possui entradas constantes O(1), porém aloca uma matriz com n*m structs, tornando sua complexidade de espaço Θ(m*n), sendo m e n a largura e altura da imagem, respectivamente.

Função Pgm::Pgm (Construtor do TAD Pgm): Por ser quase idêntica à função anterior, essa função também realiza operações constantes e possui um laço para alocação de suas colunas, tornando sua complexidade Θ(n). No caso da complexidade de espaço, por processo similar à função anterior, também é Θ(m*n)

Função readFromFile: Essa função realiza diversas atribuições, comparações e leituras de forma constante O(1) e a construção do objeto PPM O(n), porém possui dois laços aninhados para percorrer toda a matriz do TAD PPM para atribuir os valores do arquivo para as structs de pixel, que ocupa O(n * m) de complexidade de tempo. Assim, a complexidade de tempo resultante é O(1) + O(n *m), o que leva à Θ(m * n), sendo m e n a

largura e altura da imagem cujo nome é passado como parâmetro. Analisando o espaço, a função possui como entrada uma string passada por referência O(1), além de diversas outras estruturas auxiliares O(1), porém é feito a chamada para o construtor do TAD PPM, tomando O(n*m) + O(n) = O(max(n * m, n), temos que a complexidade de espaço também é Θ(n * m).

Função convertPpmToPgm: A função, possui atribuições e comparações O(1) e a chamada do construtor do TAD PGM O(n), mas também possui dois laços aninhados que percorre a matriz mxn do TAD PPM, portanto a complexidade de tempo é Θ(n * m). sendo m a largura e n a largura da imagem. Em questão ao espaço, são passados dois parâmetros O(1) (note que o TAD Ppm é um ponteiro), porém também é chamado o construtor de uma das classes, que assim como observado, toma sua complexidade como Θ(n * m).

Função writePgmToFile: Assim como as outras, a função realiza diversas atribuições e comparações O(1), mas como percorre a matriz de um dos TADs, possui complexidade de tempo Θ(m * n), sendo m a largura e n a largura da imagem. Na complexidade de espaço, são passados uma string e um TAD pgm, porém ambos são passados como referência, tornando sua complexidade de espaço total Θ(1) em vez de Θ(n * m).

4. Estratégias de Robustez

Para garantir robustez e a corretude do programa diante da entrada do usuário, foram empregadas diversas estratégias de robustez, apresentadas no formato de asserts utilizando o header “msgassert.h”.

Já iniciando pelos argumentos inseridos ao rodar o programa, há a checagem da existência do arquivo de input e output de imagem, arquivo de output do desempenho caso houver necessidade e o uso incorreto da flag do registro de memória sem a flag de desempenho. Ademais, também no arquivo Main, há asserts para verificar se os ponteiros passados como parâmetros não apontam para o nulo, assim como asserts para verificar a alocação das instâncias dos TADs alocados dinamicamente. Também há asserts que verificam se o arquivo realmente existe (FILE * não é nulo) e que está no formato correto.

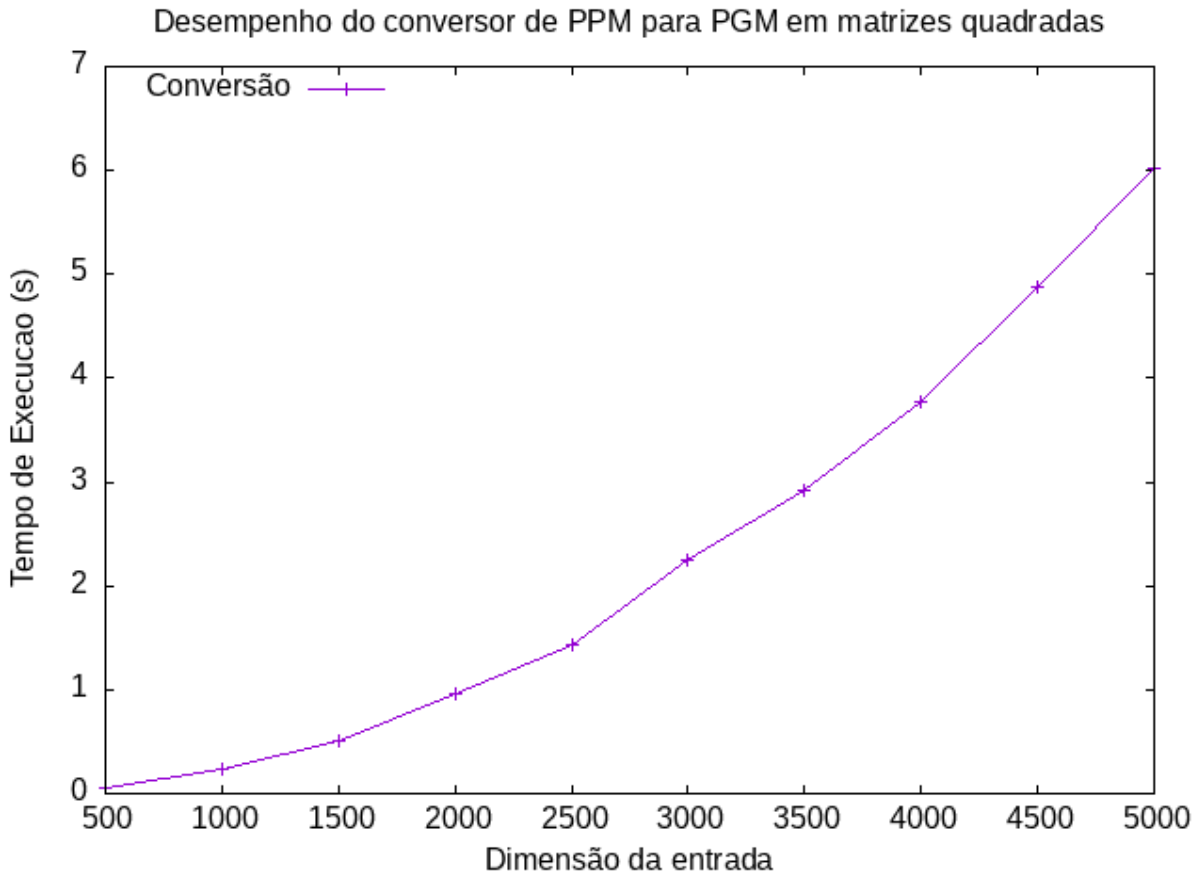
Na questão dos TADs, ambos possuem matrizes alocadas no heap, em que sempre que ocorre a verificação na alocação tanto das linhas quanto das colunas, assim como a checagem que os valores inteiros de comprimento e largura são válidos. Além disso, é importante notar que apesar do acesso dessa matriz pelas outras funções ser direto, a robustez ainda é mantida devido ao fato dos tamanhos da matriz serem atributos privados e, portanto, imutáveis após sua iniciação no construtor.

Um ponto fraco da robustez do programa seria o fato de que o arquivo PPM de entrada precisa estar corretamente formatado e válido, uma vez que esse formato de arquivo pode conter elementos como comentários ou não possuir a quantidade de pixels condizente com suas dimensões especificadas no topo dele. Também não há nenhuma verificação de que o arquivo esteja nomeado no formato .ppm, podendo ter qualquer extensão. Dessa forma, a parte relevante a critério do usuário é que seu arquivo PPM seja estruturado na forma correta.

5. Análise Experimental

Como foi visto na análise de complexidade, grande parte da complexidade de tempo das funções de manipulação dos TADs PPM e PGM possuem Θ(m * n), que varia tanto de acordo com a altura e largura da imagem. Para melhor visualização e análise do comportamento assintótico, os testes de desempenho foram realizados estritamente com

imagens quadradas, uma vez que possui um comportamento mais característico, $\Theta(n^2)$. As imagens PPM foram geradas aleatoriamente em intervalos entre 500x500 pixels e 5000x5000 pixels e o tempo de execução foi gravado para cada um dos casos.



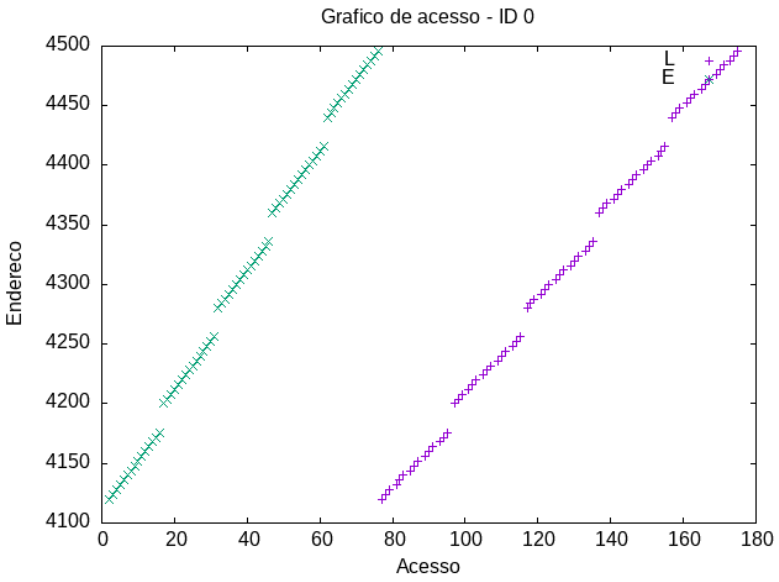
Como é possível observar, a função acima claramente demonstra uma função quadrática, o que é condizente com a análise de complexidade apresentada, uma vez que todas as funções relevantes do programa apresentam complexidade de tempo igual ou inferior a $\Theta(n*m)$, e portanto, $\Theta(n^2)$ para n = m. Assim, o gráfico reflete a complexidade "geral" do programa construído.

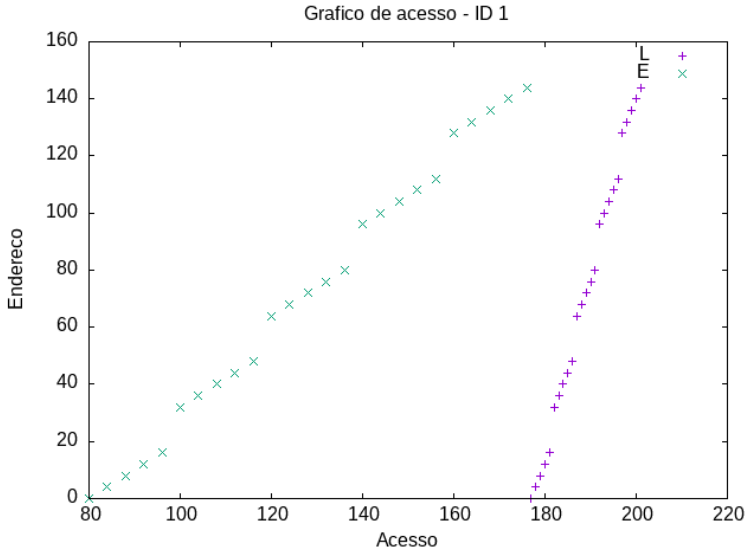
Na parte da análise de memória, optou-se pelo teste em cima de uma escala 5x5 para a melhor visualização de histogramas e distância de pilhas. Devido às escolhas de engenharia de software, as matrizes não são acessadas completamente antes das funções, o que inviabilizou a demarcação de "fases" presentes na biblioteca memlog, pois não há funções que acessam o vetor em primeira instâncias, fazendo com que os histogramas demonstram vários acessos com distância de pilha zero. Outro argumento contra essas "funções de acesso" seria a própria performance do programa. Em relação aos TADs, gravou-se a leitura e escrita dos atributos dos TADs, não sua instância diretamente. O TAD PPM recebeu o id 0 e o TAD PGM recebeu o id 1.

Após o uso do analisamem com modificações para a leitura correta dos endereços, obteve-se gráficos de acesso, evolução de distância de pilha e histogramas da distância de pilha, nos quais serão analisados um a um.

Os gráficos de acesso refletem os ciclos de leitura e escrita das duas matrizes. Em geral, no ID 0, as leituras são feitas em grupos de 15 endereços, que representam os 3 endereços de todas as 5 colunas de cada linha, que são vetores alocados dinamicamente.

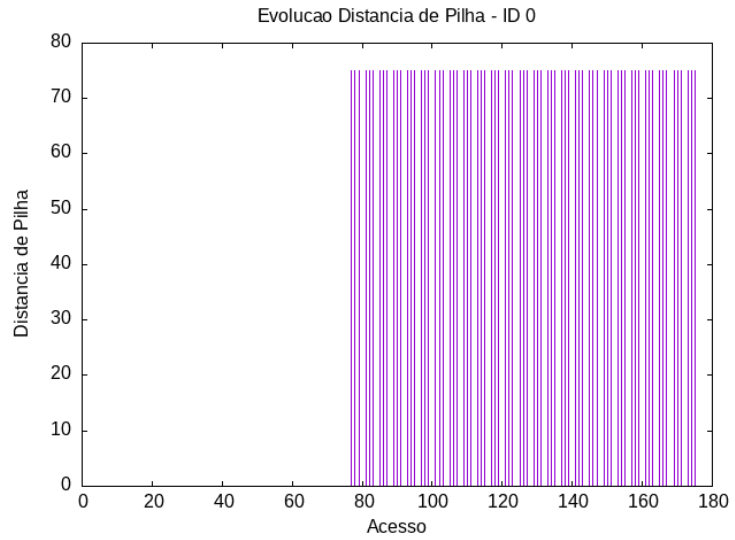
O espaçamento vertical maior entre os “blocos” representa cada um desses vetores diferentes, que representam o número de linhas da matriz. No gráfico de id 0, também há o mesmo padrão, porém há apenas 5 endereços por bloco, uma vez que há apenas um atributo (inteiro) por elemento da matriz, ao contrário do ID 0, cujos elementos são structs. O padrão de maior espaçamento horizontal durante a escrita do id 1 e leitura do id 0 é dada pelo motivo das duas operações serem feitas em conjunto durante o método convertPpmToPgm.

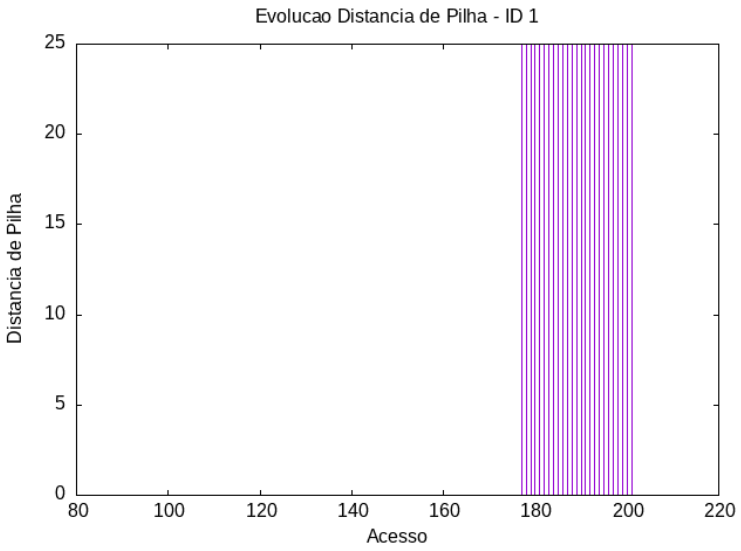




No assunto da distância de pilha, os dois gráficos seguem o mesmo padrão, mudando apenas seu comprimento novamente pela matriz do TAD PPM armazenar atributos compostos, e consequentemente, 3 endereços nos quais são escritos/lidos. No início, durante o primeiro período de escrita, há 75 ($5 * 5 * 3$) e 25 ($5 * 5$) acessos com distância de pilha 0 no id 0 e id 1, respectivamente. Pela escrita ser o primeiro acesso à variável, ela é colocada pela primeira vez na pilha, o que explica essa distância em uma primeira instância de escrita, presentes nos métodos readFromFile e convertPpmToPgm. Já

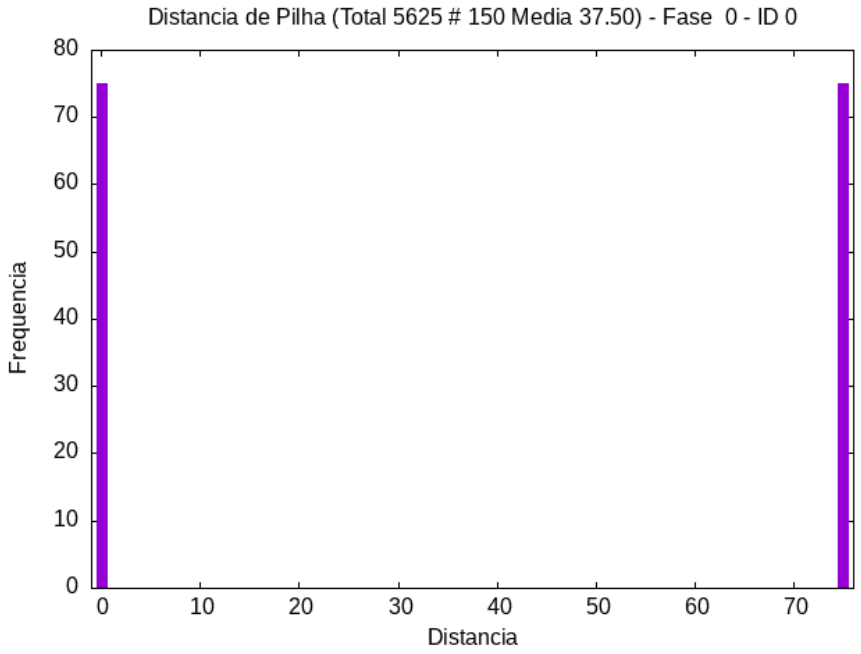
no processo de leitura, há o mesmo número de acessos com uma distância constante, porém maior que 0, que será explicada em maior detalhe na seção do histograma de pilha.

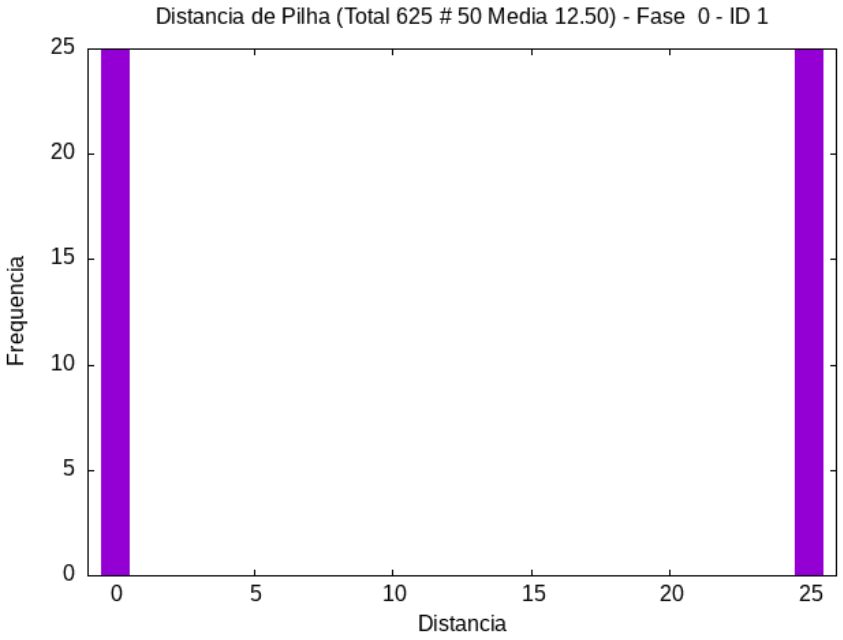




No histograma de pilha, fica nítido o padrão de acesso. Começando pelo id 1 pela clareza, há um acesso inicial para todos os elementos da matriz A_{mxn}, que se dá por $(n * m)$. Dessa forma, dada que a matriz testada é quadrada, são feitas 25 operações com distância 0. Ao percorrer a matriz em certa ordem, os primeiros elementos acessados, isto é, as colunas da primeira linha, as colunas da segunda linha, e assim em diante, são colocados no fundo da pilha, tanto que no final da primeira operação o primeiro acesso (elemento $a_{1,1}$) possui a mesma profundidade que o número total de elementos da matriz do TAD. Seguindo essa lógica, ao percorrer igualmente a matriz na próxima operação (no caso do id 1, writePgmToFile), os elementos acessados estão sempre no fundo da pilha, pois ao chamar o elemento mais profundo, o próximo a ser chamado, que era o segundo mais profundo, se torna o mais profundo, e assim em diante. Como devem ser lidos todos os elementos, há $(n * m)$ chamadas com $(n * m)$ de distância de pilha. Uma forma de reduzir essa distância seria percorrer a matriz de forma contrária à anterior, mas isso traz consigo outros custos computacionais.

De forma análoga, no id 0 também há o mesmo padrão observado no parágrafo anterior, porém de maneira levemente diferente devido aos atributos compostos utilizados. Como são utilizados 3 endereços relevantes por elemento da matriz, o número de elementos e, consequentemente, a distância de pilha máxima e o número de acessos na distância 0 (readFromFile) e na distância máxima (convertPgmToPpm) é dado por $3 * (n * m)$.





6. Conclusões

O trabalho lidou com o problema de conversão de uma imagem colorida para uma imagem em *grayscale*, exigindo a criação de TADs cujas estruturas de dados representavam matrizes, nas quais seus elementos representavam pixels das imagens, além de algoritmos envolvendo sua manipulação.

Através do desenvolvimento das funções em primeira pessoa, permitiu-se um entendimento mais aprofundado e pessoal com as análises de complexidade e tempo, reforçando a sua importância não só dentro da parte teórica, mas também como fator importante a ser levado em conta durante o desenvolvimento. Além disso, a criação dos gráficos, apesar de breve, também contribuiu fortemente para a aprendizagem na medida que serve como uma forma visual de informar o que está ocorrendo no programa de acordo com sua entrada

7. Bibliografia

Chaimowicz, L. and Prates, R. (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 para compilação e execução

Para compilar o programa, descompacte o arquivo .zip e acesse o diretório “TP”, considerado o diretório raiz do programa, com seu terminal de preferência. Logo em seguida, utilize o comando “make”.

Para executar o programa, acesse o executável gerado com o comando “./bin/run.out” com as seguintes flags:

- -i (Obrigatória): Indica no seu argumento o caminho do arquivo de entrada no formato .ppm. Um exemplo de uso seria: -i “lagoa.ppm”
- -o (Obrigatória): Indica no seu argumento o caminho do arquivo de saída do programa no formato .pgm. Note que caso já houver algum arquivo com o mesmo caminho e nome, ele será sobrescrito. Exemplo de uso: -o “lagoa.pgm”
- -p (Opcional): Habilita o output de desempenho da biblioteca memlog, tomando o caminho do arquivo gerado como seu argumento. Exemplo de uso: -p “desempenho.out”
- -l (Opcional): Deve ser usado apenas quando o output de desempenho também estiver habilitado, não admite argumentos. Habilita a análise de memória da biblioteca memlog, que é registrada também no arquivo de desempenho".

Um exemplo de uso geral seria, dado que o terminal está aberto na pasta “TP”:

./bin/run.out -i “nome ou caminho do arquivo de entrada” -o “nome ou caminho do arquivo de saída” -p “nome ou caminho do arquivo de desempenho) -l