Trabalho Prático 0

Thaís Ferreira da Silva - 2021092571

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

thaisfds@ufmg.br

1 Introdução

O problema proposto foi a implementação de um conversor de imagens, para isso, o programa deve receber uma imagem com representação de 8 bits (R,G,B com valores de 0 a 255) e extensão .ppm (ASCII, P3) e gerar uma saída em tons de cinza com representação de 2 bits (valor único de 0 a 49) .pgm (ASCII, P2).

2 Método

O programa foi desenvolvido na linguagem C++, compilado pelo compilador GCC da GNU Compiler Collection. O Computador utilizado tem as seguintes especificações:

- Sistema Operacional: Linux Ubuntu 22.04
- Processador: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz
- RAM: 8,00 GB (utilizável: 7,80 GB)

2.1 Classes

O programa é composto de duas classes principais: RGB e PPM. A classe RGB armazena os três valores correspondentes às cores primárias aditivas de uma imagem (Red, Green e Blue) e também o valor correspondente ao cinza (Gray) daquele pixel.

2.2 Funções

As classes contam com cinco funções principais: RGB, PPMreader, PPMtoPMG, e PGMwrite.

RGB: A função armazena os valores do pixel (vermelho, verde e azul) e calcula o valor da escala de cinza (variável gray) através da fórmula:

PPMreader: A função PPMreader é responsável por fazer toda a leitura do arquivo escolhido pelo usuário. Ela recebe o nome do arquivo da imagem .ppm e lê os valores iniciais do arquivo: versão, altura, largura,e valor máximo do pixel. Depois disso ela guarda os valores RGB da imagem

PPMtoPMG: A função PPMtoPMG faz a conversão da imagem de PPM para PMG através do cálculo do novo pixel em escala de cinza.

PGMwrite: A função PGMwrite é responsável por criar e escrever a nova imagem .pgm com a escala e cinza calculada. Ela recebe o nome do arquivo e através do loop vai preenchendo o arquivo.

3 Análise de Complexidade

3.1 Complexidade de tempo

Função RGB: Como a função apenas armazena os valores do RGB e calcula o valor da escala de cinza, sua complexidade é constante. Sendo assim $O(1)$.

Função PPMreader: Como a função possui um loop, que vai do 0 até o tamanho máximo do vetor para ler o arquivo .ppm e armazenar os seus valores na variável RGB, sua complexidade varia de acordo com o tamanho do vetor. Sendo assim a complexidade é O(n).

Função PPMtoPGM: Como a função possui um loop, que vai do 0 até o tamanho máximo do vetor para pegar o valor da escala de cinza e armazenar em um vetor separado, sua complexidade varia de acordo com o tamanho do vetor. Sendo assim a complexidade é $O(n)$.

Função PGMwrite: Como a função possui um loop, que vai do 0 até o tamanho máximo do vetor apenas para ler a escala de cinza e colocar no arquivo .pgm, sua complexidade varia de acordo com o tamanho do vetor. Sendo assim a complexidade é $O(n)$.

Existem outras pequenas operações ao longo do programa, mas todas possuem complexidade menor do que n (a maioria O(1)). Sendo assim:

$$O(1) + O(n) + O(n) + O(n) = O(max(1, n)) = O(n)$$

3.2 Complexidade de Espaço

Em relação ao complexidade de espaço, ao longo do programa ocorreram duas alocações dinâmicas, RGB* RGB e int* png. Como ambos são vetores de tamanho altura*largura sua complexidade é:

$$O(n) + O(n) = O(2n) = O(n)$$

4 Estratégias de Robustez

Para tornar o programa mais robusto foi utilizado do assert e uma função para verificar a extensão de um arquivo.

No main.cpp foi criado a função extensionChecker com a intenção de verificar se o final do arquivo é ppm, pgm ou algo diferente disso. Ela é utilizada para verificar se estamos lendo um arquivo no formato desejado, e se o arquivo de saída possui o final pgm. A função recebe o nome do arquivo (ex: lagoa.ppm) e a extensão desejada (.ppm), e retorna uma variável booleana.

Além disso, foram implementadas algumas medidas para analisar o nome do arquivo de entrada e o nome do arquivo de saída, para prevenir arquivos em formatos e tamanhos indesejados. Para isso foi utilizado do erroAssert no arquivo de entrada e de saída, e também um pequeno tratamento no arquivo de saída com o avisoAssert, onde o ".pgm" é adicionado ao final do nome.

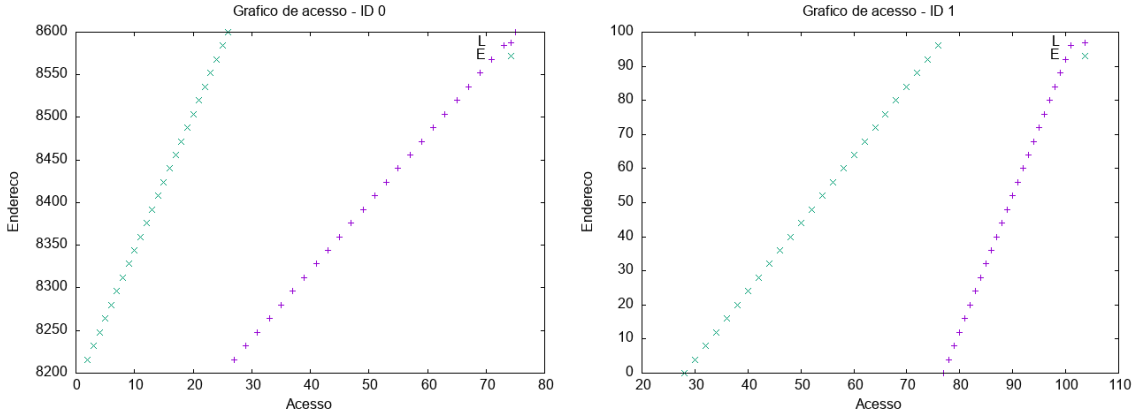
Além disso, foram implementadas algumas medidas para analisar:

- O nome do arquivo de entrada e o nome do arquivo de saída, para prevenir arquivos em formatos e tamanhos indesejados;
- A versão do arquivo de entrada (deve ser P3);

- Os valores da altura, largura e valor máximo do pixel;
- Cada valor de RGB que será armazenado;
- A abertura dos arquivos .ppm e .pgm.

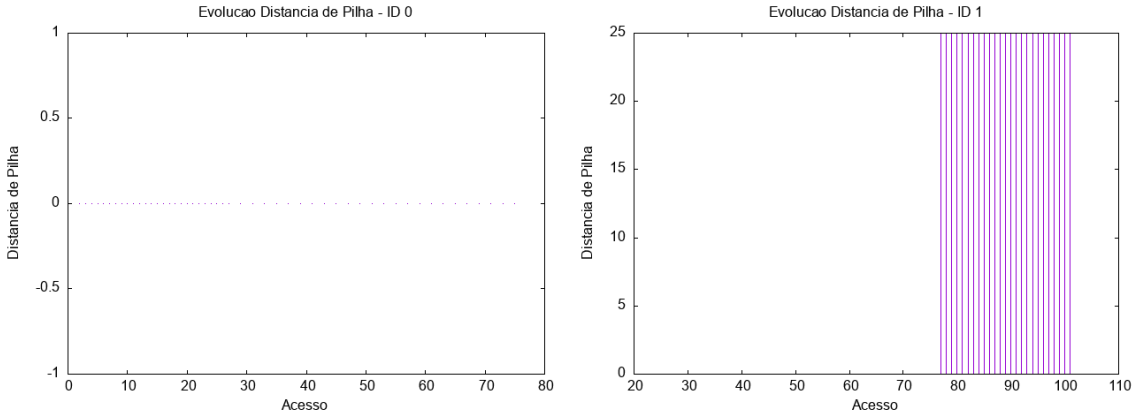
5 Análise Experimental

Através da biblioteca analisamem foi possível examinar os acessos e as distâncias de pilhas das duas variáveis alocadas dinamicamente: RGB* RGB e int* png.



(a) Acesso vetor RGB (b) Acesso vetor PGM

Figura 1: Gráfico de acesso

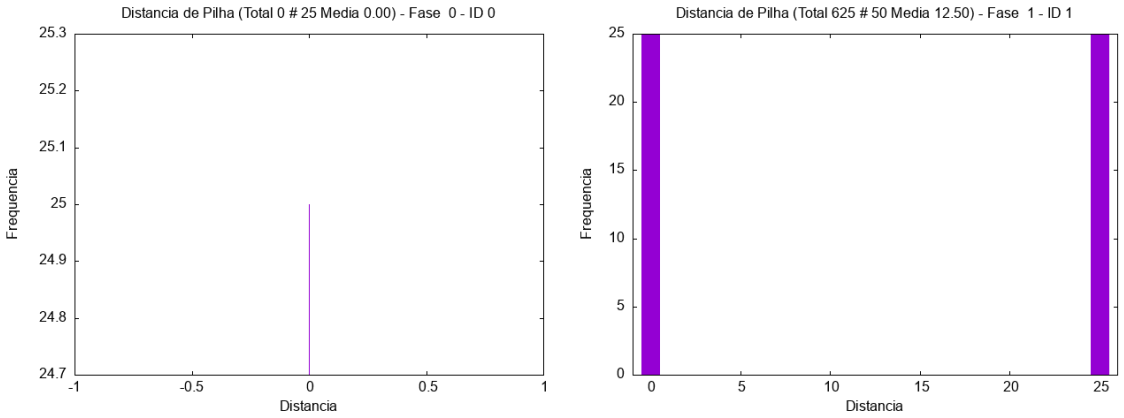


(a) Evolução Distancia de Pilha RGB (b) Evolução Distancia de Pilha PGM

Figura 2: Gráfico Evolução Distancia de Pilha

Podemos perceber através da figura 1 (a) e (b) que os acessos aos dois vetores ocorreram em dois momentos, uma para a leitura e um para a escrita. Além disso, é possível analisar através da figura 2 e 3 (a) que a evolução da distância de pilhas do vetor RGB foi constantemente 0, enquanto a figura 2 e 3 (b) mostra que houveram 25 assessores de distância de pilha 25.

Através da imagem 4 e 5 pode-se analisar o tempo de execução de cada parte do programa, onde obtemos um tempo de aproximadamente 0,04 segundos por função.



(a) Distancia de Pilha RGB (b) Distancia de Pilha PGM

Figura 3: Gráfico de Distancia de Pilha

```
 1  I 1 1239.732643932
 2  E 0 2 0 0.063856301 140101624291352 16
 3  E 0 3 0 0.063864128 140101624291368 16
 4  E 0 4 0 0.063866029 140101624291384 16
 5  E 0 5 0 0.063867452 140101624291400 16
 6  E 0 6 0 0.063868763 140101624291416 16
 7  E 0 7 0 0.063870082 140101624291432 16
 8  E 0 8 0 0.063871407 140101624291448 16
 9  E 0 9 0 0.063872707 140101624291464 16
10  E 0 10 0 0.063873986 140101624291480 16
11  E 0 11 0 0.063875531 140101624291496 16
12  E 0 12 0 0.063876582 140101624291512 16
13  E 0 13 0 0.063877481 140101624291528 16
14  E 0 14 0 0.063878435 140101624291544 16
15  E 0 15 0 0.063879429 140101624291560 16
16  E 0 16 0 0.063880403 140101624291576 16
17  E 0 17 0 0.063881317 140101624291592 16
```

Figura 4: log.out

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 20.00      0.06     0.06  1920000     0.00     0.00  RGB::RGB(int, int, int)
 20.00      0.12     0.06        1    60.00   150.00  PPM::PPMreader(std::__cxx11::basic_
 13.33      0.16     0.04        1    40.00    80.00  PPM::PPMtoPGM()
 10.00      0.19     0.03  3840000     0.00     0.00  escreveMemLog(long, long, int)
 10.00      0.22     0.03  3840000     0.00     0.00  leMemLog(long, long, int)
 10.00      0.25     0.03                             _init
  6.67      0.27     0.02  7680001     0.00     0.00  clkDifMemLog(timespec, timespec, ti
  6.67      0.29     0.02        1    20.00    40.00  PPM::PGMwrite(std::__cxx11::basic_s
  3.33      0.30     0.01  1920000     0.00     0.00  RGB::RGB()
  0.00      0.30     0.00  1920000     0.00     0.00  RGB::getGray()
  0.00      0.30     0.00  1920000     0.00     0.00  RGB::~RGB()
  0.00      0.30     0.00        3     0.00     0.00  extensionChecker(std::__cxx11::basi
  0.00      0.30     0.00        2     0.00     0.00  defineFaseMemLog(int)
  0.00      0.30     0.00        1     0.00     0.00  ativaMemLog()
  0.00      0.30     0.00        1     0.00     0.00  iniciaMemLog(char*)
  0.00      0.30     0.00        1     0.00     0.00  finalizaMemLog()
  0.00      0.30     0.00        1     0.00     0.00  __static_initialization_and_destruc
  0.00      0.30     0.00        1     0.00     0.00  __static_initialization_and_destruc
  0.00      0.30     0.00        1     0.00     0.00  __static_initialization_and_destruc
  0.00      0.30     0.00        1     0.00     0.00  PPM::PPM()
  0.00      0.30     0.00        1     0.00     0.00  __gnu_cxx::__alloc_traits<std::allc
  0.00      0.30     0.00        1     0.00     0.00  std::char_traits<char>::length(char
  0.00      0.30     0.00        1     0.00     0.00  std::allocator_traits<std::allocatc
  0.00      0.30     0.00        1     0.00     0.00  bool std::operator==<char, std::cha
  0.00      0.30     0.00        1     0.00     0.00  bool std::operator!=<char, std::cha
  0.00      0.30     0.00        1     0.00     0.00  std::__cxx11::basic_string<char, st
```

Figura 5: analise.txt

6 Conclusão

Pode-se perceber através desse trabalho como se realiza a análise de complexidade de um programa, utilizando de bibliotecas auxiliares para consolidar o resultado esperado (analisamem e memlog). Além disso, foi um ótimo exercício de revisão juntamente com o aprendizado do novo conteúdo apresentado pela disciplina de Estrutura de Dados. Devido a esses novos conhecimentos adquiridos houveram alguns desafios na elaboração do programa, como a utilização das bibliotecas de análise e a forma de se converter uma imagem PPM para PGM.

Por fim, pude através desse exercício construir um conversor de imagem colorida para escala de cinza e analisar a sua eficiência ao longo de sua execução para diversos tamanhos de arquivo.

References

SOFR. Find out if string ends with another string in C++. Stack Overflow. Disponível em: <https://stackoverflow.com/questions/874134/find-out-if-string-ends-with-another-string-in-c>. Acesso em: 20 set. 2022.

WIKIPEDIA CONTRIBUTORS. Netpbm. Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Netpbm>.
Acesso em: 20 set. 2022.

Cplusplus.com. Disponível em:
<https://cplusplus.com/reference/fstream/fstream/>.
Acesso em: 20 set. 2022.

Instruções de Compilação e Execução

1. Abrir o terminal
2. Entrar na pasta tp
   Comando: cd / <caminho para a pasta /tp >
3. Compilar o programa com o Makefile disponibilizado na pasta
   Comando: make
4. Executar o programa
   Comando: ./bin/<nome do arquivo.out> -i <nome do arquivo de entrada .ppm> -o <nome do arquivo de saída .pgm> -p <nome do arquivo de registro .out> -l
5. Após terminar de utilizar o programa, delete os arquivos desnecessários
   Comando: make clean