Conversão de imagens

Marcelo Andrade

github.com/marceloanje
linkedin.com/in/marcelo-andrade-10334b160
marcelo.anje@outlook.com

1. Introdução

Esta documentação lida com o problema de conversão de imagens, que se encontram no formato *.ppm*, para o formato *.pgm*. Por vezes, pode ser requerido a transformação dessas imagens para diversos fins, seja para diminuir o tamanho do arquivo, seja para a impressão em escala de cinza para tais.

O problema proposto foi realizar a manipulação, armazenamento e conversão de imagens no formato *.ppm*, que por sua vez, são imagens coloridas, para imagens no formato *.pgm*, que por sua vez, são em escala de cinza.

A seguinte fórmula foi utilizada na conversão das imagens:

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

2. Método

2.1 Implementação

O programa foi desenvolvido na linguagem C, compilada pelo WSL da Ubuntu e os testes foram realizados em um emulador de sistemas operacionais (VirtualBox) com Linux Ubuntu 20.04.4, sendo emulado em um computador com Windows 10, Ryzen 7 2.3 GHz e 8GB de memória RAM.

2.2 Estrutura de dados

O programa teve como base o uso de matrizes. Que são compostas por dois vetores encadeados de forma a armazenar as linhas e as colunas das matrizes. Essas matrizes são
alocadas em tempo de execução, ou seja, alocadas dinamicamente, isso significa que durante a execução do programa é escolhido na memória o tamanho exato da matriz, dessa forma, otimizando o armazenamento na memória, evitando a alocação desnecessária e um estouro de memória.

2.3 Formato de dados

O sistema foi feito de tal forma, que durante sua execução é recebido um caminho o qual deve se encontrar um arquivo de entrada no formato *.ppm*, o qual contém os dados de entradas que são tratados internamente, dessa forma a realizar todo mecanismo para

conversão da imagem. O resultado final é expelido através de um arquivo no formato *.pgm*, que pode ter seu caminho especificado através de argumentos fornecidos pelo usuário.

2.4 Classes e métodos

O programa possui uma classe, que é a struct ***mat***, onde se encontram as definições e o escopo dos métodos usados. Além dessas três classes, também há o arquivo ***main.c***.

A struct ***mat*** possui possui cinco métodos, que são: *criaMatriz*, que é usado para criar uma matriz alocada dinamicamente, ainda sem valores, *inicializaValores*, que é usado para ler o arquivo *.ppm* e inserir os valores em uma matriz, *converteValores*, que é usado para converter os valores em *.ppm* para valores em *.pgm*, *criaImagemPGM*, que é usado para criar um arquivo do formato *.pgm* e *destroiMatriz* que é usado para desalocar uma matriz.

O arquivo ***main.c*** é onde ocorre a execução do programa em si, essa parte recebe o arquivo de entrada fornecido pelo usuário e o destino do arquivo de saída. Essa parte utiliza os métodos para realizar a leitura, conversão e escrita dos arquivos.

3. Análise de complexidade

A análise de complexidade do programa foi analisada os métodos relevantes da classe ***mat***. Dessa forma, sendo analisada no âmbito do tempo e espaço.

3.1 Complexidade temporal

Função criaMatriz - esta função realiza um ‘for’ para alocar dinamicamente a matriz - O(n);
Função inicializaValores - esta função utiliza dois ‘for’ aninhados para preencher a matriz e aciona o método *criaMatriz*, que tem ordem de complexidade O(n), sendo assim, suas ordens de complexidade são somadas - O(n^2);
Função converteImagem - esta função utiliza dois ‘for’ aninhados para preencher a matriz e aciona o método *criaMatriz*, que tem ordem de complexidade O(n), sendo assim, suas ordens de complexidade são somadas - O(n^2);
Função criaImagemPGM - esta função utiliza dois ‘for’ aninhados para imprimir a matriz - O(n^2);
Função destroiMatriz - possui apenas alguns atribuições - O(1);

Como a ordem de complexidade do programa todo é a soma das ordens de todas as funções, a ordem de complexidade do programa todo é O(n^2)

3.2 Complexidade espacial

Para a complexidade espacial podemos avaliar o principal método de armazenamento de dados do programa, que é a matriz. A matriz usada no programa é uma matriz de duas dimensões, ou seja n elementos das linhas vezes n elementos das colunas, dessa forma podemos considerar que o espaço ocupado é O(n^2), também podemos analisar que essa

ordem de complexidade não irá se alterar, independe do número de matrizes no espaço, pois *c*O(f(n)) = O(f(n))*.

4. Estratégias de robustez

As funções do programa, são tratadas para prevenção de erros de manipulação de matriz com o *erroAssert* que quando ocorre uma não conformidade retorna uma mensagem de erro.

Os erros não toleráveis se dizem respeito à manipulação das matrizes, como verificar se o tamanho especificado é um tamanho válido, ou seja, maior que zero, ou verificar se uma matriz já foi desalocada.

A prevenção de erros ocorre utilizando o *erroAssert* também na abertura de arquivos, que retorna uma mensagem de erro se o arquivo for invalido.

Portanto, o tratamento de erros ocorre nas funções, por meio de mensagens de erro, deixando claro ao usuário caso haja algum problema.

5. Conclusão

Esse trabalho teve como objetivo a implantação de um sistema de conversão de imagens, a partir disso foi desenvolvido um programa em C para atender a todas as especificações propostas.

Com esse desenvolvimento foi possível reforçar bem alguns conceitos, como o de matrizes e alocação dinâmica.

Referências

Ziviani, N., **Projeto de Algoritmos com Implementações em Pascal e C**, 3ª Edição, Cengage Learning, 2011.

Instruções de compilação e execução

Para a execução do programa sigas os passos descritos abaixo, utilizando o Makefile para isso:

- Utilizando o terminal acesse o diretório;
- Execute o arquivo Makefile utilizando o seguinte comando: “make”;
- Execute o *main* no terminal ainda no diretório, escrevendo “programa -i “caminho de entrada” -o “arquivo de saída""
- Para apagar todos os arquivos criados na compilação, execute no terminal o comando: “make clean”;