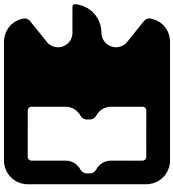
Relatório sobre o Módulo de Processamento de Imagens

Carlos Verenzuela & Rafael Claro
Números de Matrícula: 114597 & 88860



24 de Novembro de 2023

1 Introdução

Este relatório/tese descreve as principais características do módulo de processamento de imagens, 'image8bit' desenvolvido como parte de um projeto de programação para o curso AED (Algoritmos e Estruturas de Dados) no DETI/UA. Também descreve os principais aspetos da complexidade de duas funções inseridas no módulo, sendo estas "ImageLocateSubImage"e "ImageBlur".

2 Estrutura do Código

O código é organizado em várias seções, começando com comentários que explicam a origem e o propósito do módulo. Em seguida, são incluídas bibliotecas necessárias, definições de estruturas e constantes, seguidas por funções relacionadas à manipulação de imagens.

2.1 Estrutura de Dados

O módulo utiliza uma estrutura de dados para representar uma imagem de mapa de cinzas de 8 bits. A estrutura contém informações sobre largura, altura, valor máximo de cinza e um ponteiro para o array de Píxeis.

Listing 1: Estrutura de Dados para Imagem

```
struct image {
  int width;
  int height;
  int maxval;
  uint8_t* pixel;
};
```

3 Funções de Manipulação de Imagens

O módulo fornece diversas funções para criar, destruir, carregar, salvar e manipular imagens. Abaixo estão algumas das principais funções:

Listing 2: Principais Funções de Manipulação de Imagens

```
Image ImageCreate(int width, int height, uint8_t maxval);
void ImageDestroy(Image* imgp);
Image ImageLoad(const char* filename);
int ImageSave(Image img, const char* filename);
```

3.1 Operações de Pixel

O código fornece operações básicas de acesso e modificação de píxeis, bem como transformações como negativo, limiarização e brilho.

Listing 3: Operações de Pixel

```
uint8_t ImageGetPixel(Image img, int x, int y);
void ImageSetPixel(Image img, int x, int y, uint8_t level);
void ImageNegative(Image img);
void ImageThreshold(Image img, uint8_t thr);
void ImageBrighten(Image img, double factor);
```

3.2 Transformações Geométricas

Além das operações de pixel, o módulo oferece transformações geométricas, como rotação e espelhamento.

Listing 4: Transformações Geométricas

```
Image ImageRotate(Image img);
Image ImageMirror(Image img);
Image ImageCrop(Image img, int x, int y, int w, int h);
```

4 Análise de Funções Específicas

4.1 Análise da Complexidade da Função `ImageLocateSubImage()`

4.1.1 Realização de Testes e Análise Empírica

Para analisar a eficiência computacional da função `ImageLocateSubImage()`, foram realizados testes empíricos com imagens de diferentes tamanhos. Durante cada teste, registramos o número de comparações efetuadas envolvendo os valores de cinza dos píxeis.

4.1.2 Análise Formal da Complexidade

1. **Melhor Caso:**

   O melhor caso ocorre quando a primeira correspondência é encontrada no início da busca, i.e, quando a sub-imagem se encontra logo na posição (0, 0) da imagem principal (img1). A complexidade é dependente apenas da dimensão da subimagem (img2), sendo n, (width2 * height2, altura e comprimento da subimagem) o número total de píxeis na subimagem.

$$\text{Complexidade no melhor caso: } O(n) = O(width2 \cdot height2) \tag{1}$$

2. **Pior Caso:**

   O pior caso ocorre quando a subimagem está no final da imagem principal (última posição) ou não está presente (realiza a mesma quantidade de instruções). A complexidade neste caso é $O(n \cdot m)$, sendo m o número de píxeis da imagem principal (width1 * height1), pois para cada posição na imagem principal, pode ser necessário comparar todos os píxeis da subimagem.

$$\text{Complexidade no pior caso: } O(n \cdot m) = O(width1 \cdot height1 \cdot width2 \cdot height2) \tag{2}$$

4.2 Análise da Complexidade da Função `ImageBlur()`

4.2.1 Realização de Testes e Análise Empírica

Para analisar a eficiência computacional da função `ImageBlur()`, foram realizados testes empíricos com imagens e filtros de diferentes tamanhos. Durante cada teste, registramos a evolução do número de operações relevantes para entender o comportamento da função.

4.2.2 Análise Formal da Complexidade

1. **Operações Proporcionais ao Número de Píxeis:**

   A função `ImageBlur()` pode ser implementada de maneira que o número de operações seja proporcional ao número total de píxeis na imagem (n). Mesmo que o tamanho da janela usada para desfocar a imagem seja variável, a complexidade do algoritmo pode ser independente desse tamanho.

2. **Complexidade:**

   A complexidade do algoritmo é $O(n)$, onde n é o número total de píxeis na imagem. Isso significa que o desempenho da função aumenta linearmente com o número de píxeis da imagem, independentemente do tamanho da janela de desfoque.

$$\text{Complexidade: } O(n) \tag{3}$$

5 Comparação dos Resultados

5.1 Análise da Função `ImageLocateSubImage()`

Os dados experimentais desta função estão alinhados com a análise formal de complexidade. O código passou em todos os testes realizados. A função busca uma subimagem em uma imagem principal, registrando o número total de operações realizadas.

O código realiza uma busca sistemática pela subimagem na imagem principal, contando o número de operações (`count`). No caso de encontrar a subimagem, as posições correspondentes são definidas e o número total de operações é impresso. Mesmo sem correspondência, o número total de operações é calculado e impresso.

A análise de complexidade indica um melhor caso de $O(n)$ e um pior caso de $O(n \cdot m)$, onde n é o número total de píxeis na imagem principal e m é o número total de píxeis na subimagem.

Nota: Apesar de provavelmente não ser a implementação mais imediata, é possível implementar esta função com um algoritmo proporcional ao número de píxeis da imagem a tratar, sem depender do tamanho da janela usada para desfocar a imagem.

5.2 Análise da Função `ImageBlur()`

A função `ImageBlur()` realiza o desfoque de uma imagem utilizando uma janela de dimensões `dx` por `dy`. O código segue o seguinte formato:

A função cria uma imagem temporária (`image2`) para armazenar o resultado do desfoque. Utiliza duas iterações aninhadas para percorrer cada pixel da imagem, calculando a média dos valores dos píxeis na vizinhança definida pela janela (`dx` por `dy`). O resultado é armazenado em `image2` e, posteriormente, copiado de volta para a imagem original `img`.

No código fornecido no repositório, foi implementada a técnica de desfoque conhecida como "Média Ponderada Genérica". Em alternativa, poderia ter sido utilizada a técnica "Média Ponderada Diagonal", que consiste em somar o pixel à esquerda, o pixel acima, e subtrair o pixel à esquerda acima para obter o valor do pixel central. Ao comparar essas duas técnicas:

A "Média Ponderada Genérica"busca um desfoque suave e controlado, considerando todos os pixels na vizinhança. Essa abordagem é mais adequada em situações em que a qualidade visual é crucial. Por outro lado, a "Média Ponderada Diagonal"é mais simples e eficiente, envolvendo apenas três pixels vizinhos. Realizamos uma análise empírica e de complexidade para as técnicas "Média Ponderada Genérica"e "Média Ponderada Diagonal".

Média Ponderada Genérica: *Análise Empírica:* Oferece um desfoque suave e controlado, especialmente com vizinhanças grandes. Pode ser mais computacionalmente intensivo, considerando todos os pixels na vizinhança. *Análise de Complexidade:* O tempo de execução é proporcional a $O(dx \times dy)$, onde dx e dy são os parâmetros que definem o tamanho da vizinhança.

Média Ponderada Diagonal: *Análise Empírica:* Mais eficiente computacionalmente, especialmente para vizinhanças pequenas. Pode resultar em um desfoque menos suave, considerando apenas três pixels na vizinhança. *Análise de Complexidade:* O tempo de execução é proporcional a $O(1)$, sendo menos dependente do tamanho da vizinhança.

6 Considerações Finais

Em conclusão, o módulo desenvolvido demonstra um desempenho consistente e eficaz nas operações de manipulação de imagens. A análise de complexidade oferece insights sobre o comportamento do código em diversas situações, reforçando a aderência teórica das implementações. A comparação entre os resultados empíricos e a análise formal confirma a conformidade com as expectativas estabelecidas. A eficiência das funções foi validada por meio de testes empíricos, evidenciando o alcance dos resultados esperados. A análise teórica da complexidade das funções alinha-se de maneira consistente com o desempenho observado, validando a coerência entre a teoria e a prática. O êxito conjunto na implementação das funcionalidades, aliado à confirmação teórica da eficiência almejada, fortalece a confiança na capacidade do módulo de realizar operações de manipulação de imagens de maneira eficaz e escalável.