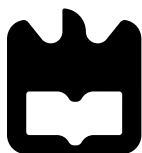


AED - Projeto 1

Universidade de Aveiro

Orlando Marinheiro, Tomás Oliveira, Afonso Vieira



AED - Projeto 1

Dept. de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

afonso.vieira@ua.pt(112822), tomas.esteves.oliveira@gmail.com(113939), orlandomarinheiro@ua.pt(114060)

26 de novembro de 2023

Conteúdo

1	Introdução	1
2	Análise	2
2.0.1	Análise ImageLocateSubImage	2
2.0.2	Análise ImageBlur	4
3	Conclusão	5

Capítulo 1

Introdução

Este relatório apresenta uma análise detalhada relativa à eficiência e qualidade das funções `ImageLocateSubImage()` e `ImageBlur()`. A análise dessas funções foi realizada com o propósito de compreender o desempenho dos algoritmos implementados bem como o número de operações realizadas.

O objetivo principal foi realizar sequências de testes com imagens de diferentes tamanhos, avaliando diversos cenários de entrada de modo a entender o comportamento dos algoritmos propostos.

Para atingir esse objetivo, foram seguidas certas metodologias para cada função em questão. Para a função `ImageLocateSubImage()`, foram realizados testes que analisaram o número de comparações em imagens de tamanhos distintos. Além disso, foi realizada uma análise formal da complexidade para os casos melhor e pior.

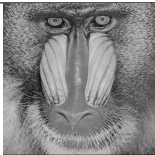

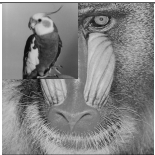
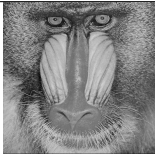

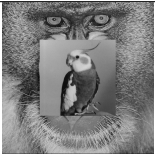
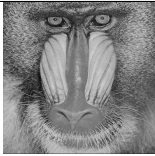











Na função `ImageBlur()`, os testes foram conduzidos também com diferentes tamanhos de imagens, considerando a evolução do número de operações relevantes para o desfoque da imagem bem como uma análise formal da complexidade do algoritmo.

Este relatório é dividido em seções dedicadas a cada função, apresentando tabelas com os resultados obtidos nos testes realizados. A comparação entre os resultados das diferentes abordagens e estratégias utilizadas é discutida para proporcionar uma visão abrangente do desempenho de cada função.

Capítulo 2

Análise

2.0.1 Análise ImageLocateSubImage

Imagem (img1)	Subimagem (img2)	Imagem + Subimagem(Paste)	(x,y)	Nº de comparações
			(0,0)	65536
			(127,127)	98283
			(255,255)	131502
			(0,0)	65536
			(671,471)	702207
			(1343,943)	1339603

A partir da realização dos testes anteriores verificamos que com o aumento do tamanho da imagem, o número de comparações aumenta. Para além disso, à medida que as coordenadas (x,y) variam de posição, isto é, quando estão localizadas no canto superior esquerdo, centro e canto inferior direito, também podemos verificar que o número de comparações aumenta para ambos os tamanhos das imagens.

Passando à análise formal do melhor e pior caso, podemos concluir que o melhor caso é quando as coordenadas (x,y) são respetivamente (0,0), ou seja, quando a subimagem se localiza no canto superior esquerdo, sendo o seu valor igual à área da subimagem. Posto isto, a ordem de complexidade do melhor caso pode ser representado pela seguinte fórmula:

$$O(Width_Img2 \times Height_Img2)$$

Já o pior caso acontece quando todos os pixels da imagem(img1) têm que ser comparados, neste caso até chegar ao último pixel de img1, (que se encontra no canto inferior direito), como mostra o seguinte esquema.

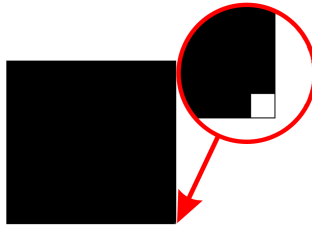


Figura 2.1: Demonstração do último pixel da imagem a ser iterado

Posto isto, relativamente à ordem de complexidade do pior caso, temos que:

$$O((Width_Img1 - Width_Img2) \times (Height_Img1 - Height_Img2) \times Width_Img2 \times Height_Img2)$$

Já que obtivemos as fórmulas do pior e melhor caso, através destas é possível calcular o número de operações substituindo as dimensões das imagens nas respetivas variáveis. Por exemplo, img1 tem dimensões **512x512** e img2 tem dimensões **256x256**. Colocando a subimagem(img2) nas coordenadas (0,0) de img1 e aplicando a fórmula para o melhor caso, obtemos o seguinte:

$$O(Width_Img2 \times Height_Img2) = 256 \times 256 = 65536$$

```

Loading pgm/small/bird_256x256.pgm -> I0
Loading tests_ImageLocateSubImage/paste2_1.pgm -> I1
Locating I0 in I1
Número de comparações: 65536
# FOUND (0,0)

```

Outro exemplo é, duas imagens com cor preta sólida, img1 com dimensões **512x512** e img2 com dimensões **256x256**, com um pixel branco no canto inferior direito. Colocando a subimagem(img2) nas coordenadas (255,255) de img1 e aplicando a fórmula para o pior caso, obtemos o seguinte:





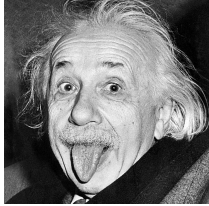
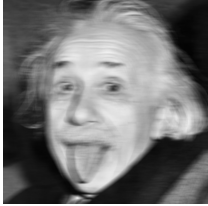
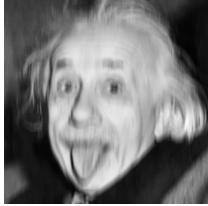

$$O((Width_Img1 - Width_Img2) \times (Height_Img1 - Height_Img2) \times Width_Img2 \times Height_Img2) \\ = (512 - 256) \times (512 - 256) \times 256 \times 256 = 4294967296$$

```

Loading extra_tests/fff_1.pgm -> I0
Loading tests_ImageLocateSubImage/paste1_4.pgm -> I1
Locating I0 in I1
Número de comparações: 4294967296
# FOUND (255,255)

```

2.0.2 Análise ImageBlur

Imagem Original	Blur (20,3)	Blur (3,20)	Blur (20,20)
			
Nº de Operações	36263776	36263776	203444768
			
Nº de Operações	506045920	506045920	2911570400

A partir da realização dos testes da função ***ImageBlur(dx,dy)*** verificamos que o número de operações aumenta consoante o aumento do tamanho da imagem. No caso dos testes realizados anteriormente, primeiramente atribuímos a dx e dy os valores 20 e 3 respetivamente para ambas as imagens. No segundo teste invertemos os valores de dx e dy, podendo concluir que o número de operações é exatamente o mesmo em ambos os casos. Por último definimos dx e dy com o valor 20 e verificamos um aumento significativo do número de operações nas duas imagens.

Passando à análise formal do algoritmo, é possível obter a fórmula da ordem de complexidade desta função.

```
// Percorrer todos os pixels dessa imagem
for (int j = 0; j < height; j++) {
    for (int i = 0; i < width; i++) {
        int count = 0; // Inicialização de uma variável de contagem de pixels
        int soma = 0;  // Inicialização de variavel da soma de pixels
```

Na imagem acima, é fácil de entender que as duas iterações resultam na ordem de complexidade $Width \times Height$.

```
for (int x = -dx; x <= dx; x++) { // Percorrer pixels entre as posições (x-dx, x+dx)
    for (int y = -dy; y <= dy; y++) { // Percorrer pixels entre as posições (y-dy, y+dy)
```

Posteriormente, estas duas iterações mostram que x e y variam de -dx até dx e -dy até dy, respetivamente. Logo na primeira iteração a ordem de complexidade é $2 \times dx + 1$ e na segunda é $2 \times dy + 1$. Assim, juntando tudo e como temos 4 iterações seguidas, é necessário multiplicar ambas as fórmulas obtidas anteriormente, originando a seguinte ordem de complexidade da ***ImageBlur(dx,dy)***:

$$O(Width \times Height \times (2 \times dx + 1) \times (2 \times dy + 1))$$

Capítulo 3

Conclusão

A análise da complexidade das funções `ImageLocateSubImage()` e `ImageBlur()` ofereceu perspectivas valiosas sobre a eficiência desses algoritmos. Ao realizar alguns testes e análises, podemos perceber como o desempenho dessas funções é sensível em diferentes aspectos, como os tamanhos das imagens, a intensidade dos pixels e os filtros utilizados.

Em suma, este projeto para além de nos ter proporcionado uma compreensão mais aprofundada sobre a eficiência destas funções, também realçou a importância de estratégias algorítmicas e da consideração cuidadosa dos parâmetros de entrada para o desempenho eficaz dos algoritmos de processamento de imagens.