Algoritmos e Estruturas de Dados

2023/2024 — 1º Semestre

1º Trabalho — O TAD image8bit

Ricardo Martins (112876)

Tiago Brito (112911)

Índice

Introdução

Este relatório apresenta a análise de eficiência computacional de duas funções críticas implementadas no âmbito do Tipo Abstrato de Dados (TAD) `image8bit`. As funções em foco são `ImageLocateSubImage()` e `ImageBlur()`. A primeira destina-se a localizar uma subimagem em uma imagem maior, enquanto a segunda aplica um filtro de desfoque à imagem.

A avaliação da eficiência dessas funções é essencial para compreender como elas se comportam em diferentes cenários e tamanhos de imagens. Este relatório detalhará a realização de testes práticos com diversas imagens, analisando o número de operações envolvendo os valores de cinzento dos pixels.

Além dos testes práticos, realizaremos uma análise formal da complexidade para ambas as funções, considerando os melhores e piores casos.

Análise Experimental ImageLocateSubImage()

Melhor Caso: O(2*(H2xW2))

O melhor caso ocorre quando a sub imagem encontrada está na posição (0,0) da imagem principal, tendo a função ImageMatchSubImage apenas de percorrer os pixéis da subimagem, de dimensões W2xH2, enquanto realiza duas operações de GetPixel, logo terá uma complexidade igual a O(2*(H2*W2))

Sendo o menor número possível de operações igual a 2*(1*1) = 2, para o caso particular em que a imagem é apenas de um pixel que se encontra na posição (0,0) da imagem principal tendo a função ImageMatchSubImage de verificar apenas a posição (0,0).

test_BestCase	Img1	Img2	Position	Count
Test1	White_10x10	White_1x1	(0,0)	2
Test2	White_100x100	White_1x1	(0,0)	2
Test3	White_1000x1000	White_1x1	(0,0)	2
Test4	White_10000x10000	White_1x1	(0,0)	2
Test5	White_10x10	White_10x10	(0,0)	200
Test6	White_10000x10000	White_10x10	(0,0)	200
Test7	White_100x100	White_100x100	(0,0)	20000
Test8	White_10000x10000	White_100x100	(0,0)	20000

Ricardo Martins
Tiago Brito

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Pior Caso: O(2*((W1-W2 + 1) * (H1-H2 + 1) * (W2 * H2))

O pior caso ocorre quando a sub imagem não está presente na imagem principal ou está na última posição possível, sendo a função ImageMatchSubImage chamada para cada posição possível na imagem principal, fazendo 2 operações de ImageGetPixel sempre que é chamada.

Assim sendo suponhamos que a imagem maior (img1) tem dimensões W1×H1 e a sub imagem (img2) tem dimensões W2×H2, a função vai usar dois loops aninhados para percorrer todas as posições possíveis da sub imagem dentro da imagem maior. Portanto, a complexidade temporal seria proporcional ao número de iterações desses dois loops. Ou seja a complexidade seria O(2*((W1-W2 + 1) * (H1-H2 + 1) * (W2 * H2))

A análise experimental será feita usando como img2 apenas um pixel, de forma a garantir que é possível testar a função para imagens1 de dimensões cada vez maiores sem que haja overflow do número total de operações executadas.

Test_WorstCase	Img1	Img2	Position	Count
Test1	White_10x10	Black_1x1	NA	200
Test2	White_100x100	Black_1x1	NA	20000
Test3	White_1000x1000	Black_1x1	NA	2000000
Test4	White_10000x10000	Black_1x1	NA	200000000

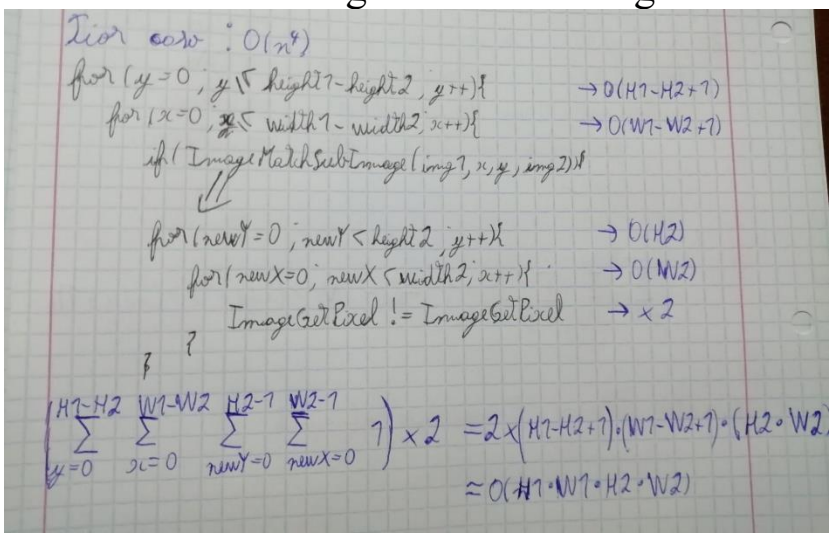
```
test_ImageLocate : $(PROGS) setup
    ./imageTool createWhite 10,10 save White_10x10.pgm
    ./imageTool createWhite 100,100 save White_100x100.pgm
    ./imageTool createWhite 1000,1000 save White_1000x1000.pgm
    ./imageTool createWhite 10000,10000 save White_10000x10000.pgm
    ./imageTool create 1,1 save Black_1x1.pgm
    ./imageTool create 10,10 save Black_10x10.pgm

    ./imageTool Black_1x1.pgm White_10x10.pgm paste 5,5 save testLocate1.pgm
    ./imageTool Black_1x1.pgm White_1000x1000.pgm paste 5,5 save testLocate2.pgm
    ./imageTool Black_10x10.pgm White_1000x1000.pgm paste 5,5 save testLocate3.pgm
    ./imageTool Black_10x10.pgm White_1000x1000.pgm paste 50,50 save testLocate4.pgm
    ./imageTool Black_10x10.pgm White_1000x1000.pgm paste 500,500 save testLocate5.pgm

    ./imageTool Black_1x1.pgm testLocate1.pgm locate
    ./imageTool Black_1x1.pgm testLocate2.pgm locate
    ./imageTool Black_10x10.pgm testLocate3.pgm locate
    ./imageTool Black_10x10.pgm testLocate4.pgm locate
    ./imageTool Black_10x10.pgm testLocate5.pgm locate
```

	Img1	Img2	Position	BestCase < Count< WorstCase
Test1	White_10x10	Black_1x1	(5,5)	2 < 112 < 200
Test2	White_1000x1000	Black_1x1	(5,5)	2 < 10012 < 2000000
Test3	White_1000x1000	Black_10x10	(5,5)	200 < 10120 < 196416200
Test4	White_1000x1000	Black_10x10	(50,50)	200 < 198600 < 196416200
Test5	White_1000x1000	Black_10x10	(500,500)	200 < 992200 < 196416200

Ricardo Martins
Tiago Brito

deti universidade de aveiro departamento de electrónica, telecomunicações e informática

Análise Formal ImageLocateSubImage

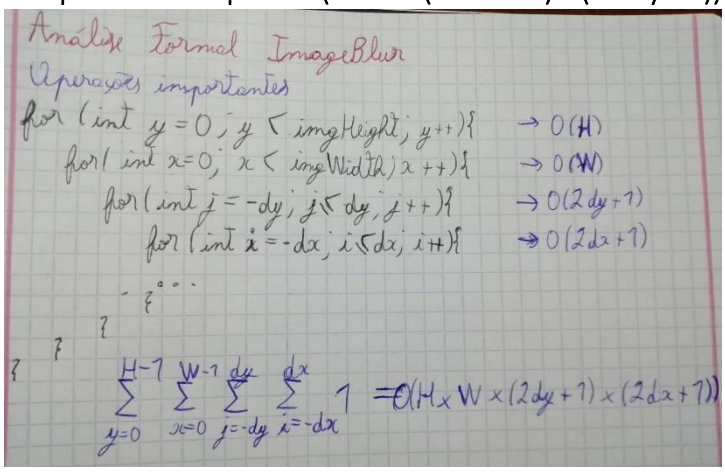


Análise Formal ImageBlur()

A complexidade temporal da função ImageBlur depende principalmente do número de iterações no loop aninhado, que percorre cada pixel da imagem.

Sendo W a largura da imagem e H a sua altura a fórmula da complexidade temporal será:

Complexidade temporal: O(W* H * (2 * dx + 1) * (2 * dy + 1))



Análise Formal ImageBlurOptimized:

1. Dentro da função ComputeIntegralImage é necessário alocar memória para todos os pixeis da imagem que estamos a criar - O(W*H)

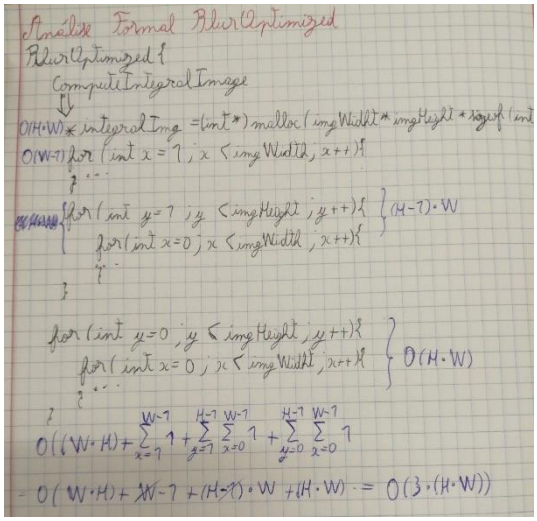
2. Depois fazemos dois loops para iterar sobre todos os pixeis da imagem – O(W*H)

3. Por fim dentro da função ImageBlurOptimized voltamos a iterar sobre todos os pixeis da imagem para alterar os seus valores de acordo com o valor fornecido para o blur

Assim sendo temos uma complexidade temporal de: **O(3*(imgWidth * imgHeight))**

Ricardo Martins
Tiago Brito

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática



Análise Experimental ImageBlur / ImageBlurOptimized

```
test_Blur: $(PROGS) setup
    ./imageTool createRandom 10,10 save Random_10x10.pgm
    ./imageTool createRandom 100,100 save Random_100x100.pgm
    ./imageTool createRandom 1000,1000 save Random_1000x1000.pgm
    ./imageTool createRandom 10000,10000 save Random_10000x10000.pgm
    ./imageTool Random_10x10.pgm blur 33,3
    ./imageTool Random_10x10.pgm blurOptimized 33,3
    ./imageTool Random_100x100.pgm blur 33,3
    ./imageTool Random_100x100.pgm blurOptimized 33,3
    ./imageTool Random_1000x1000.pgm blur 33,3
    ./imageTool Random_1000x1000.pgm blurOptimized 33,3
    ./imageTool Random_10000x10000.pgm blur 33,3
    ./imageTool Random_10000x10000.pgm blurOptimized 33,3
    ./imageTool Random_1000x1000.pgm blur 8,5
    ./imageTool Random_1000x1000.pgm blurOptimized 8,5
    ./imageTool Random_1000x1000.pgm blur 80,5
    ./imageTool Random_1000x1000.pgm blurOptimized 80,5
```

TestCases	value	Blur(real)	Blur(Formula)	BlurOptimized
Random_10x10	33,3	6200	46900	300
Random_100x100	33,3	3877664	4690000	30000
Random_1000x1000	33,3	464355464	469000000	3000000
Random_10000x10000	33,3	47213433464	overflow	300000000
Random_1000x1000	8,5	189700160	187000000	3000000
Random_1000x1000	80,5	1699084400	1771000000	3000000

Conclusão

Este relatório conclui o desenvolvimento e teste do Tipo Abstrato de Dados (TAD) `image8bit` e a análise de eficiência computacional das funções `ImageLocateSubImage()` e `ImageBlur()`. Realizamos testes práticos para ambas as funções, registando o número de operações relevantes, e conduzimos análises formais de complexidade para compreender seu desempenho teórico.

Em resumo, este trabalho destaca a implementação eficaz do TAD `image8bit` e fornece uma análise detalhada da eficiência das funções.

Ricardo Martins
Tiago Brito

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática