Algoritmos e Estruturas de dados

TAD image8bit

Abel José Enes Teixeira nº 113655
Filipe Pina de Sousa nº114196
Novembro 2023

Prof. Mário Antunes
Prof. Regente Joaquim Madeira

Ano letivo
2023-2024

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

1 Introdução

Este trabalho tem como objetivo completar,desenvolver,modificar funções para: criar, apagar, copiar imagens, visualizando resultados tais como: *time, caltime, pixmem, comparisons* com ajuda do *imageTool.c* em que cada pixel pode ter um valor de intensidade entre 0 e 255 (8 bits) em tons de branco a preto.

É também pedido a analise da complexidade assim como registar as comparações de duas funções: **ImageLocateSubImage()** e **ImageBlur()**.

ImageLocateSubImage() é uma função que caso exista a localização determina uma subimagem numa imagem escolhida pelo usuario,localiza a posição de uma determinada subimagem (img2) dentro de uma imagem maior (img1) se não existir e retornada a mensagem *Not Found*.

ImageBlur() é uma função que aplica um filtro com um nivel que o usuario decide para uma imagem a sua escolha e a torna baça.

2 ImageLocateSubImage()

Esta função serve para localizar uma imagem noutra de tamanho igual ou superior. A função **ImageLocateSubImage()** chama outra função, a **ImageMatchSubImage()**, que compara a imagem de tamanho inferior com uma porção da imagem de tamanho superior.

2.1 Análise formal

A análise formal serve para analisar a complexidade de um código transformando-o numa fórmula matemática e desenvolvendo-a.

Complexidade

A complexidade pode ser dividida em duas partes principais: a iteração pelos pixels da img1 e a chamada da função `ImageMatchSubImage()` para comparar as subimagens.

A iteração é feita no seguinte código:

```
for (int y = 0; y <= img1->height - img2->height; y++){
    for (int x = 0; x <= img1->width - img2->width; x++){
```

O loop externo itera pelas linhas, enquanto o loop interno itera pelas colunas da img1.

```
if (ImageMatchSubImage(img1, x, y, img2))
```

Dentro dos loops, a função `ImageMatchSubImage()` é chamada para comparar as subimagens.

```
for (int i = 0; i < img2->height; i++) {
        for (int j = 0; j < img2->width; j++) {
```

O Melhor caso de um Algoritmo é quando ocorre menor número de iterações.
O Pior caso de um Algoritmo é quando ocorre o maior número de iterações.

Melhor Caso: Ambas as imagens tem cada pixel igual verificando logo na primeira posição:

$$\sum_{y1=0}^{h2-1}\sum_{x1=0}^{w2-1} 1 = \sum_{y=0}^{h2-1} w2 = h2 \cdot w2 \tag{1}$$

Pior Caso: A img1 e maior e ser tudo em preto e a img2 e mais pequena,completamente preta mas com um pixel que é diferente da cor preta no final da imagem ou encontrar apenas no final.

$$\begin{aligned}
&\sum_{y=0}^{h1-h2}\sum_{x=0}^{w1-w2}\sum_{y1=0}^{h2-1}\sum_{x1=0}^{w2-1} 1 = \\
&= \sum_{y=0}^{h1-h2}\sum_{x=0}^{h2-1}\sum_{y1=0}^{h2-1} w2 \\
&= \sum_{y=0}^{h2-h1}\sum_{x=0}^{w1-w2} h2 \cdot w2 \\
&= \sum_{y=0}^{h2-h1} (w1 - w2 + 1) \cdot h2 \cdot w2 \\
&= (h1 - h2 + 1) \cdot (w1 - w2 + 1) \cdot h2 \cdot w2
\end{aligned} \tag{2}$$

2.2 Análise Experimental

A análise experimental serve para obter dados a partir de vários *inputs* do utilizador servindo assim para comprovar a análise formal.

tipo de teste	img1	dim1	img2	dim2	comparações	Output
img1 < img2	Bird	256x256	original	300x300	0	Not Found
diferentes	Einstein	940x940	original	300x300	411096	Not Found
cortada	mandrill	512x512	mandrillcut1	25x50	100471	Found
mesma	Einstein	940x940	Einstein	940x940	883600	Found
pior caso	Black	1000x1000	PixelBranco	64x64	3596161024	Not Found

3 ImageBlur()

A função **ImageBlur()** aplica um *mean filter* de

$$[2dx + 1] \cdot [2dy + 1]$$

em imagens do formato **.pgm**. O filtro torna a imagem baça, a intensidade varia consoante os valores de ***dx*** e ***dy***.

3.1 Algoritmo 1

O primeiro algoritmo utiliza uma abordagem de força bruta, onde são percorridos todos os pixeis e todos os valores do filtro, calculando a soma em cada um deles.

3.1.1 Análise formal

Complexidade:

No algoritmo 1 do ImageBlur() o pior e o melhor caso são iguais pois ele tem de percorrer todos os pixeis da imagem, ou seja a sua complexidade será igual à seguinte equação:

$$\begin{aligned}
&\sum_{x=0}^{w-1}\sum_{y=0}^{h-1}\left(1+\sum_{x=-dx}^{dx}\sum_{y=-dy}^{dy}1\right) = \qquad (3)\\
&= \sum_{x=0}^{w-1}\sum_{y=0}^{h-1}1+\sum_{x=0}^{w-1}\sum_{y=0}^{h-1}\sum_{x=-dx}^{dx}\sum_{y=-dy}^{dy}1\\
&= (w\cdot h)+\left(w\cdot h\cdot\sum_{x=-dx}^{dx}\sum_{y=-dy}^{dy}1\right)\\
&= (w\cdot h)+(w\cdot h\cdot(2dx+1)\cdot(2dy+1))
\end{aligned}$$

3.1.2 Análise experimental

Como podemos visualizar na tabela de testes, este algoritmo perde eficiência quanto maior o filtro e a imagem a ser filtrada, sendo inutilizável para filtros maiores.

tipo de teste	tamanho	dx	dy	time	calltime	pixmem
enunciado	300x300	7	7	0.058183	0.089534	20340000
pequeno	256x256	10	10	0.082949	0.124946	28966912
		50	50	1.901882	2.937568	668598272
		150	150	16.831131	25.864553	5937692672
média	512x512	10	10	0.331882	0.502090	115867648
		50	50	7.601877	11.624807	2674393088
		150	150	67.679453	104.051100	23750770688
grande	940x940	10	10	1.116112	1.714742	390551200
		50	50	25.717143	38.606396	9014487200
		150	150	227.627423	350.890256	80055927200

Table 1: Tabela de resultados para o algoritmo 1

3.2 Algoritmo 2

O segundo algoritmo usa uma abordagem mais eficiente que o primeiro, utilizando uma tabela de soma acomulada para realizar menos acessos a memória.

3.2.1 Análise formal

Complexidade:

No algoritmo 2 do ImageBlur() o melhor e pior caso também são iguais a complexidade do mesmo,segue a seguinte equação:

$$\begin{aligned} &\sum_{x=0}^{w+2\cdot dx-1} \sum_{y=0}^{h+2\cdot dy-1} 1 + \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} 1 \\ =&(w + 2 \cdot dx) \cdot (h + 2 \cdot dy) + w \cdot h \end{aligned} \tag{4}$$

3.2.2 Análise experimental

Como podemos visualizar na seguinte tabela de testes, este algoritmo é muito mais eficiente porque acede muito menos vezes à memoria *pixmem* e a variação do tempo que demora a correr quando se aumenta o filtro é insignificante, fazendo o tamanho da imagem seja a única variável a alterar o tempo de aplicar o filtro que o código demora a aplicar o filtro.

tipo de teste	tamanho	dx	dy	time	calltime	pixmem
enunciado	300x300	7	7	0.000751	0.001148	188596
pequeno	256x256	10	10	0.000586	0.000901	141712
		50	50	0.000827	0.001271	192272
		150	150	0.001576	0.002414	374672
médio	512x512	10	10	0.002037	0.003094	545168
		50	50	0.004683	0.006853	636688
		150	150	0.004197	0.006760	921488
grande	940x940	10	10	0.007895	0.012056	1805200
		50	50	0.008482	0.012971	1965200
		150	150	0.010786	0.016608	2421200

Table 2: Tabela de resultados para o algoritmo 2

4 Conclusão

Este trabalho consolidou os conhecimentos adquiridos durante o semestre sobre conceitos como implementação e avaliação de algoritmos na linguagem de programação C, *Latex, Valgrind, Git* e como testar o nosso Trabalho.

Para verificar a eficiência do nosso código, foi utilizada como teste a funcionalidade *tic toc*, que faz *reset* aos contadores todos e ao tempo quando é introduzido *tic*, o *toc* permite-nos visualizar os contadores e o tempo entre a execução do código, ambos passados via linha de commandos antes de executar o código.

Neste trabalho, durante a execução do algoritmo 1, procedemos a melhorias progressivas, resultando na formulação do algoritmo 2. Este aprimorou as fragilidades do algoritmo anterior, incorporando conceitos de otimização que foram consolidados ao longo do semestre.