universidade de aveiro

TAD IMAGE8BIT

ALGORITMOS E ESTRUTURAS

DE DADOS 2023

Prof. Mário Antunes

José Marques / 114321 ½ (50%)

João Gaspar / 114514 ½ (50%)

Índice

Introdução

Na cadeira de Algoritmos e Estruturas de Dados foi-nos proposto o desenvolvimento de um Tipo Abstrato de dados que representa uma imagem PGM. Neste relatório, será descrito como foi desenvolvido as funções **ImageLocateSubImage** e **ImageBlur**. Isso inclui a descrição dos algoritmos utilizados, funcionamento do programa e a sua análise formal. Além disso, são apresentados os resultados obtidos ao testar o programa com diferentes entradas e discutidas as suas conclusões.

Funções

Este capítulo, irá incidir sobre as funções desenvolvidas durante o desenvolvimento deste projeto. Assim, será explicado as funcionalidades das funções, a sua análise formal e ainda, a apresentação de dados, testes e resultados.

Função ImageMatchSubImage()

A função **ImageMatchSubImage** compara uma subimagem (img2) com uma parte específica de uma imagem maior (img1) a partir de uma posição definida (x, y). Ela verifica se as imagens são não nulas, valida a posição inicial na imagem maior e realiza a comparação de pixels. Se encontrar pixels diferentes, a função retorna 0, indicando falta de correspondência; caso contrário, retorna 1, indicando uma correspondência bem-sucedida.

Função ImageLocateSubImage()

A função **ImageLocateSubImage** tenta encontrar a posição de uma subimagem (img2) dentro de uma imagem maior (img1). Assim, recorrendo a dois ciclos for e a um if statement, ela vai examinar todas as possíveis posições na imagem maior, chamando a função **ImageMatchSubImage** para verificar correspondência em cada posição. Se encontrar uma correspondência, ela armazena as coordenadas e retorna 1. Se nenhuma correspondência for encontrada, a função retorna 0.

Análise Formal

Como objeto de estudo de performance, iremos definir os acessos à memória como indicativo de complexidade de todas as funções em análise, dado que é a operação mais densa, assim iremos desenvolver uma análise formal onde calcularemos uma solução fechada para o cálculo do número total de acessos à memória. Na função LocateSubImage temos 2 ciclos for encadeados, que percorrerão a imagem até ao limite de onde a subimagem poderá estar, que é no ponto (w1-w2, h1-h2), ou até haver uma correspondência. Em cada ciclo será acedido a função MatchSubImage, que por sua vez irá percorrer, através de dois ciclos for encadeados, a subimagem até encontrar uma diferença nos pixéis.

Best Case

No que consta o melhor caso da função **ImageLocateSubImage**, será o caso em que o programa encontra a subimagem no início da imagem (Fig1).

$$\left(\sum_{i=0}^{\text{width}-1}\sum_{j=0}^{\text{height}-1}(2)\right) = \left(2\sum_{i=0}^{\text{width}-1}(\text{height})\right) = 2\cdot\text{width}\cdot\text{height}$$

Fig1: Análise formal do melhor caso da função ImageLocateSubImage

Worst Case

O pior caso para a função **ImageMatchSubImage** ocorre quando a subimagem (img2) pertence à imagem maior (img1), exceto por um único pixel no canto inferior direito. Isso implica que a

função terá que percorrer toda a imagem maior para encontrar essa diferença, já que ela compara cada pixel correspondente entre as duas imagens. O desempenho da função atinge o seu pior caso quando a diferença está no último pixel a ser verificado (Fig2).

- Minorante do pior caso: A subimagem pertence à imagem, mas está na posição (w_1- w_2, h_1- h_2). Para a realização foi necessário a subtração um dois, devido à soma dos somatórios incluir duas vezes o acesso à memoria de (w1-w2, h1-h2).

$$\left(\sum_{x=0}^{w_1-w_2}\sum_{y=0}^{h_1-h_2}(2)\right)+\left(\sum_{i=0}^{w_2-1}\sum_{j=0}^{h_2-1}(2)\right)-2$$
$$=\left(2\cdot\sum_{x=0}^{w_1-w_2}(h_1-h_2+1)\right)+\left(2\cdot\sum_{i=0}^{w_2-1}(h_2)\right)-2$$
$$=2\cdot((w_1-w_2+1)(h_1-h_2+1)+w_2\cdot h_2-1)$$

Fig2: Análise Formal do minorante do pior caso da função ImageLocateSubImage

- Majorante do pior caso: O número máximo de acessos à memoria que a função poderá fazer, caso tenha de verificar em cada ponto se a subimagem pertence até ao último pixel, mas nunca pertencendo (Fig3).

$$\left(\sum_{x=0}^{w_1-w_2}\sum_{y=0}^{h_1-h_2}\sum_{i=0}^{w_2-1}\sum_{j=0}^{h_2-1}(2)\right)=\left(2\cdot\sum_{x=0}^{w_1-w_2}\sum_{y=0}^{h_1-h_2}\sum_{i=0}^{w_2-1}(h_2)\right)$$
$$=\left(2\cdot\sum_{x=0}^{w_1-w_2}\sum_{y=0}^{h_1-h_2}(w_2\cdot h_2)\right)=\left(2\cdot\sum_{x=0}^{w_1-w_2}((h_1-h_2+1)\cdot w_2\cdot h_2)\right)$$
$$=2\cdot(w_1-w_2+1)(h_1-h_2+1)\cdot w_2\cdot h_2$$

Fig3: Análise Formal do majorante do pior caso da função ImageLocateSubImage

Dados/Testes

Como podemos verificar pelos testes, temos o dobro de comparações que temos de acessos à memória, tal pode se comprovar aquando da leitura do código da função. Também podemos comparar que os valores de acessos à memória são exatamente os valores esperados no cálculo da análise formal (Tabela1 Tabela2).

Tipo	Tamanho Img1	Tamanho Imagem 2	Tempo	Caltime	Pixmen	Comparates	Análise Formal
Large	940x940	50x50	0.000008	0.000005	5000	2500	5000
	1600x1200	500x500	0.001142	0.000759	500000	250000	500000
	1200x1600	200x250	0.001130	0.000752	500000	250000	100000
	640x480	100x200	0.000061	0.000041	40000	20000	40000

Medium	512x512	20x20	0.000002	0.000001	800	400	800
	765x460	50x150	0.000023	0.000015	15000	7500	15000
	640x480	640x480	0.000003	0.0.0002	1600	800	1600
Small	222x217	10x10	0.000001	0.000000	200	100	200
	300x300	60x50	0.000012	0.000008	6000	3000	6000
	256x256	20x40	0.000003	0.000002	1600	800	1600

Tabela1: Tabela do melhor caso da função ImageLocateSubImage.

Tamanho Img1	Tamanho Img2	Tempo	Caltime	Pixmen	Comparates	Análise Formal
940x940	50x50	0.004003	0.002655	1592760	796380	1592760
1600x1200	500x500	0.004817	0.003192	2043600	1021800	2043600
1200x1600	200x250	0.007147	0.003438	2804700	1402350	2804700
640x480	100x200	0.000866	0.000439	344040	172020	344040
512x512	20x20	0.001357	0.000715	486896	243448	486896
765x460	50x150	0.001235	0.000544	460350	230175	460350
640x480	640x480	0.001167	0.000623	614400	307200	614400
222x217	10x10	0.000261	0.000137	88806	44403	88806
300x300	60x50	0.000347	0.000219	126980	63490	126980
256x256	20x40	0.000347	0.000163	104456	52228	104456

Tabela2: Tabela do pior caso da função ImageLocateSubImage.

Função ImageBlur()

A função **ImageBlur** realiza um blur na imagem de entrada (img) utilizando um método de média ponderada. Dois parâmetros, dx e dy, especificam as dimensões do kernel de blur para definir a vizinhança. Primeiro, é criada uma imagem temporária (tempImg) para armazenar o resultado. A função então percorre cada pixel na imagem original, calcula a média ponderada dos valores dos pixels na vizinhança determinada por dx e dy, e atribui esse valor ao pixel correspondente na imagem temporária. Finalmente, os valores da imagem temporária são copiados de volta para a imagem original, concluindo o processo de blur.

No entanto, este processo possui diversos problemas, tais como, a sua performance que fica dependente do tamanho da janela.

Função ImprovedImageBlur()

A função **ImageBlurImproved** aprimora o algoritmo de desfoque em uma imagem (img). Inicialmente, ela aloca dinamicamente um array chamado level_sum para armazenar somas cumulativas dos valores dos pixels. De seguida, durante duas iterações, a função vai calcular essas somas, tanto na horizontal quanto na vertical. A última etapa utiliza a soma cumulativa

para realizar um desfoque mais eficiente, evitando cálculos redundantes. Após o processo de desfoque, o array level_sum é libertado da memória.

Deste modo, resolve o problema da implementação anterior, pois, evita acessos redundantes à memória.

Análise Formal

Esta análise formal do **ImageBlur** básico assume que o número de acessos à memória na janela do filtro mantém-se constante, o que não é verdade como acontece no pixel inicial, no entanto, tal é feito para simplificar as contas (Fig4).
Assim, no Imageblur temos 2 ciclos for encadeado, que percorrem cada ponto da imagem, onde temos outros dois ciclos for que percorrem a janela do filtro de dimensões (2dx+1) * (2dy+1) seguido de 1 acesso à memoria.

$$\left(\sum_{x=0}^{\text{width}-1}\sum_{y=0}^{\text{height}-1}\left(1+\sum_{i=x-dx}^{x+dx}\sum_{j=y-dy}^{y+dy}(1)\right)\right)+\left(\sum_{x=0}^{\text{width}-1}\sum_{y=0}^{\text{height}-1}(2)\right)$$
$$=\left(\sum_{x=0}^{\text{width-1}}\sum_{y=0}^{\text{height-1}}\left(1+\sum_{i=x-dx}^{x+dx}(2dy+1)\right)\right)+\left(2\cdot\sum_{x=0}^{\text{width-1}}\text{height}\right)$$
$$=\left(\sum_{x=0}^{\text{width-1}}\sum_{y=0}^{\text{height}-1}(1+(2dx+1)(2dy+1))\right)+(2\cdot\text{width-1}\cdot\text{height-1})$$
$$=\left(\sum_{x=0}^{\text{width-1}}(\text{height}\cdot(1+(2dx+1)(2dy+1)))\right)+(2\cdot\text{width}\cdot\ \text{height}\ )$$
$$=(\text{width}\cdot\text{height}\cdot(1+(2dx+1)(2dy+1)))+(2\cdot\text{width}\cdot\ \text{height}\ )$$
$$=(\text{width}\cdot\text{heigth})\cdot(3+(2dx+1)(2dy+1))$$

Fig4: Análise Formal da função ImageBlur.

ImageBlur melhorado: Nesta versão do **ImageBlur**, identificamos que, de facto, o número de acessos à memória não depende, de todo, do tamanho da janela do filtro, dependendo apenas do tamanho da imagem, o que diminui consideravelmente número de acessos totais à memória (Fig5).

Deste modo, na função temos 4 ciclos for, do qual os 3 ultimos são duplos, o primeiro percorre a primeira coluna da imagem fazendo 2 acessos à memoria por ciclo. O segundo percorre todas as linhas excepto os da primeira coluna com 3 acessos à memoria por ciclo. O terceiro percorre todas as colunas ignorando os da primeira linha, com 3 acessos à memoria por ciclo. Por fim iremos percorrer todos os pixéis para o cálculo final e acedendo à memoria 5 vezes em cada iteração.

$$\left(\sum_{y=0}^{\text{height}-1}(2)\right)+\left(\sum_{y=0}^{\text{height}-1}\sum_{x=1}^{\text{width}-1}(3)\right)+\left(\sum_{x=0}^{\text{width}-1}\sum_{y=1}^{\text{height}-1}(3)\right)+\left(\sum_{y=0}^{\text{height}-1}\sum_{x=0}^{\text{width}-1}(5)\right)$$
$$=(2\cdot\text{height})+\left(\sum_{y=0}^{\text{height}-1}(3\cdot(\text{width}-1))\right)+\left(\sum_{x=0}^{\text{width-1}}(3\cdot(\text{height}-1))\right)+\left(\sum_{y=0}^{\text{height}-1}(5\cdot\text{width})\right)$$
$$=(2\cdot\text{height})+(3\cdot(\text{width}-1)(\text{height}))+(3\cdot(\text{height}-1)(\text{width}))+(5\cdot\text{width}\cdot\text{height})$$
$$=\text{height}(11\cdot\text{width}-1)-3\cdot\text{width}$$

Fig5: Análise Formal da função ImprovedImageBlur.

Dados/Testes

Como podemos observar pelos resultados dos testes, o nosso blur melhorado tem uma performance significativamente melhor do que o algoritmo de brute-force. Podemos também concluir, e comprovar pela análise formal, que a segunda função não depende da janela do filtro, depende apenas do tamanho da imagem. Por fim, os resultados obtidos nestes testes comprovam os cálculos da análise formal (Tabela3 Tabela4).

Tipo	Tamanho	Filtro	Tempo	Caltime	Pixmen	Análise Formal
Large	940x940	13x21	1.956670	1.308631	1009512484	1028510400
	1600x1200	5x12	1.098644	0.732979	530119080	533760000
	1200x1600	40x40	23.288433	15.498233	12233617600	12602880000
Medium	640x480	10x10	0.289124	0.192375	133821700	136396800
	512x512	20x20	0. 837276	0.557082	423993616	441450496
	765x460	50x40	5.002632	3.344495	2662404000	2879949600
	640x480	30x30	2.050004	1.369511	1081340100	1144012800
Small	222x217	10x10	0.046560	0.029839	20387266	21389256
	300x300	60x50	0.523882	0.348589	269470800	1100160000
	256x256	20x40	0.25998	0.25298	192607904	217841664

Tabela3: Tabela da função ImageBLur.

Tipo	Tamanho	Filtro	Tempo	Caltime	Pixmen	Análise Formal
Large	940x940	50x50	0.011054	0.007393	9715840	9715840
	1600x1200	5x12	0.023239	0.015504	21114000	21114000
	1200x1600	40x40	0.024923	0.016586	21114800	21114800
Medium	640x480	10x10	0.003705	0.002466	3376800	3376800
	512x512	20x20	0.003400	0.002262	2881536	2881536
	765x460	50x40	0.003760	0.002514	3868145	3868145
	640x480	30x30	0.003428	0.002290	3376800	3376800
Small	222x217	10x10	0.000508	0.000325	529031	529031
	300x300	60x50	0.000951	0.000633	988800	988800
	256x256	20x40	0.000741	0.000493	719872	719872

Tabela4: Tabela da função ImprovedImageBlur.

CONCLUSÃO

Concluindo, a solução para este problema revelou-se bem-sucedida, uma vez que alcançamos os resultados espectáveis. Ainda, para a resolução deste projeto, foi necessária uma extensa pesquisa, e, no final, adquirimos conhecimentos valiosos.

Que nos proporcionou a oportunidade de aplicar os conhecimentos adquiridos ao longo do semestre na cadeira, assim como adquirir conhecimentos valiosos para a área de algoritmia.