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

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

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).

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

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

```
jpmarques@jpmarques-Victus-by-HP-Laptop-16-d1xxx:~/Projeto_AED/trabalho1-114321-
114514$ ./imageTest pgm/large/einstein_940x940.pgm out3test.pgm
# Image Locate Melhor caso (Img2 = 50x50)
#          time         caltime          pixmem        Compares
       0.000008        0.000005            5000            2500
# Image Locate Pior caso (subimage = 50,50)
#          time         caltime          pixmem        Compares
       0.004587        0.003093         1813800          906900
```

Fig4: Tabela do melhor e pior caso para subimagem de dimensões 50x50

```
jpmarques@jpmarques-Victus-by-HP-Laptop-16-d1xxx:~/Projeto_AED/trabalho1-114321-
114514$ ./imageTest pgm/medium/airfield-05_640x480.pgm out3test.pgm
# Image Locate Melhor caso (Img2 = 100x200)
#          time         caltime          pixmem        Compares
       0.000061        0.000041           40000           20000
# Image Locate Pior caso (subimage = 1000x300)
#          time         caltime          pixmem        Compares
       0.003163        0.002134         1283100          641550
```

Fig5: Tabela do melhor e pior caso para subimagem de dimensões 100x200 e 1000x300 respetivamente

```
jpmarques@jpmarques-Victus-by-HP-Laptop-16-d1xxx:~/Projeto_AED/trabalho1-114321-
114514$ ./imageTest pgm/medium/airfield-05_640x480.pgm out3test.pgm
# Image Locate Melhor caso (Img2 = 640x480)
#          time         caltime          pixmem        Compares
       0.001035        0.000698          614400          307200
# Image Locate Pior caso (subimage = 1000x100)
#          time         caltime          pixmem        Compares
       0.002465        0.001662         1002800          501400
```

Fig6: Tabela do melhor e pior caso para subimagem de dimensões 640x480 e 1000x100 respetivamente

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 (Fig4 Fig5 Fig6).

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 (Fig7).

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

Fig7: 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 (Fig8).

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

Fig8: Análise Formal da função Improved ImageBlur

Dados/Testes

```
jpmarques@jpmarques-Victus-by-HP-Laptop-16-d1xxx:~/Projeto_AED/trabalho1-114321-
114514$ ./imageTest pgm/large/airfield-05_1600x1200.pgm out9.pgm
# Blur image
#           time         caltime          pixmem
       1.948978        1.201707       846024100
# Improved Blur image
#           time         caltime          pixmem
       0.027428        0.016912        21114000
```

Fig9: Tabela da aplicação das funções de blur a uma imagem do tipo large

Reduz o acesso em 824910100, ou seja, em 4006%.

```
jpmarques@jpmarques-Victus-by-HP-Laptop-16-d1xxx:~/Projeto_AED/trabalho1-114321-
114514$ ./imageTest pgm/medium/mandrill_512x512.pgm out6.pgm
# Blur image
#           time         caltime          pixmem
       0.249917        0.162927       114038596
# Improved Blur image
#           time         caltime          pixmem
       0.003525        0.002298         2881536
```

Fig10: Tabela da aplicação das funções de blur a uma imagem do tipo medium

Reduz o acesso em 111157060, ou seja, em 3957%

```
jpmarques@jpmarques-Victus-by-HP-Laptop-16-d1xxx:~/Projeto_AED/trabalho1-114321-
114514$ ./imageTest pgm/large/einstein_940x940.pgm out1.pgm
# Blur image window 10x10
#           time         caltime          pixmem
       0.823477        0.554576       387987700
# Improved Blur image window 10x10
#           time         caltime          pixmem
       0.009838        0.006625         9715840
# Blur image window 30x30
#           time         caltime          pixmem
       5.999058        4.040104      3184738900
# Improved Blur image window 30x30
#           time         caltime          pixmem
       0.009583        0.006454         9715840
```

Fig11: Tabela da aplicação das funções blur a imagens com diferentes filtros

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 (Fig9 Fig10 Fig11).

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.