universidade de aveiro
theoria poiesis praxis

Departamento de Eletrónica, Telecomunicações e Informática

Unidade Curricular de Algoritmos e Estrutura de Dados

2023/2024 – 1.º semestre

Relatório de Análise de Funções: ImageLocateSubImage() e ImageBlur()

26 de novembro de 2023

Docente: Prof. Dr. Pedro Lavrador

André Gonçalves nº. 99203 (andrerg@ua.pt)

Bruno Pereira nº. 112726 (brunoborlido@ua.pt)

Índice

1. Função ImageLocateSubImage()

1.1. Sequência de testes

Tabela 1.1. Resultados dos testes para ImageLocateSubImage

Imagem (pixels)	SubImagem (pixels)	Número de Comparações
300x300	200x200	837,700
300x300	100x100	528,700
300x300	50x50	401,200
800x600	500x500	2,600,844
800x600	400x400	4,093,938
800x600	300x300	5,540,532
800x600	200x200	6,437,832
800x600	100x100	115,171,788

1.2. Análise formal da complexidade do algoritmo

Complexidade de tempo aproximadamente O((w1 w2 + 1) (h1 h2 + 1) w2 h2), em que w1 e h1 são as dimensões da imagem principal, e w2 e h2 são as dimensões da subimagem.

- Melhor caso: Ocorre quando a subimagem é encontrada na primeira posição da imagem maior. Relativamente à complexidade O(1), o "O" é utilizado para descrever o comportamento assintótico do tempo de execução, sendo que neste caso, ele é constante independentemente do tamanho de entrada, fazendo assim o número mínimo de comparações necessárias.

- Pior caso: Ocorre quando a subimagem não é encontrada em nenhuma posição da imagem maior. Em termos de complexidade O((w1 - w2 + 1) · (h1 - h2 + 1) · w2 · h2), onde w1, h1 são as dimensões da imagem maior e w2, h2 são as dimensões da subimagem.

- Comparação: Enquanto a sequência de testes fornece insights práticos sobre o desempenho da função em diferentes cenários, a análise formal da complexidade oferece uma visão abstrata do comportamento do algoritmo. A função recorre a dois loops interligados para percorrer todas as posições possíveis da subimagem na imagem maior. Dentro desses loops, há outro conjunto de loops para comparar pixels. Comparando a sequência de testes obtida (Tabela 1.1) e a análise formal da complexidade do algoritmo, podemos observar que o algoritmo desenvolvido abrange os dois cenários mais extremos possíveis, o melhor e o pior caso. Com base nos resultados obtidos, no melhor caso possível, o algoritmo tem um tempo de execução constante independentemente das dimensões das imagens, enquanto que, no pior caso possível, o algoritmo desenvolve as comparações necessárias,

demorando o tempo necessário, admitindo assim, um número de comparações máximo diferente para cada imagem, de forma que possa ser mais preciso e eficiente.

2. Função ImageBlur()

A função ImageBlur() realiza um desfoque na imagem utilizando um filtro de tamanho definido.

2.1. Sequência de testes

Tabela 2.1. Resultados dos testes para ImageBlur()

Imagem (pixels)	Filtro (pixels)	Número de Operações
300x300	7x7	20,109,136
300x300	10x10	38,676,100
300x300	20x20	141,494,400
300x300	30x30	302,076,900
800x600	7x7	108,747,136
800x600	10x10	210,378,100
800x600	20x20	784,868,400
800x600	30x30	1,709,442,900

2.2. Análise formal da complexidade do algoritmo

A complexidade de tempo depende principalmente do tamanho da imagem (número de pixels) e do tamanho do filtro (dimensões do filtro). Se a implementação for realizada de forma eficiente, a complexidade pode ser proporcional ao número de pixels na imagem (N), onde N é o número total de pixels na imagem.

2.2.1. Estratégia convencional

Complexidade $O(N * k * k)$ onde N é o número total de pixels na imagem e k é o tamanho do filtro.

2.2.2. Estratégia otimizada

Uma implementação eficiente pode ser alcançada com recurso a técnicas como a convolution ou média ponderada. A convolution é uma operação que requer a multiplicação e a soma dos valores do pixel e dos pixéis que estão à sua volta, contidos no espaço do filtro. Esta técnica utiliza operações de matrizes ou técnicas de processamento de sinal, o que permite ser aplicada de forma eficiente. No caso de a implementação ser otimizada desta forma, a complexidade pode ser O(N), onde N é o número total de pixels na imagem.

2.2.3. Comparação de estratégias

Ao comparar as diferentes estratégias algorítmicas para a função ImageBlur(), foi possível observar que a estratégia otimizada com recurso à técnica convolution apresenta um desempenho melhorado em comparação à estratégia convencional, sendo que existe uma redução na complexidade do algoritmo, resultando em tempos de execução menores, especialmente para imagens grandes e filtros grandes.

3. Conclusão

Após a análise formal da complexidade das funções em causa (ImageLocateSubImage() & ImageBlur()), podemos observar que a complexidade do algoritmo da função ImageLocateSubImage() depende principalmente do tamanho da imagem e da subimagem, sendo então fundamental considerar essa complexidade ao lidar com imagens de grande escala. Em relação à função ImageBlur(), esta pode ser otimizada para uma complexidade linear (O(N)), com recurso a técnicas como a convolution. Esta abordagem oferece uma melhoria significativa em termos de eficiência computacional.

Assim sendo, e após o trabalho desenvolvido, consideramos que a escolha da estratégia algorítmica é crucial para o desempenho das funções relacionadas com o processamento de imagens. Este relatório fornece, assim, uma visão geral das análises de complexidade e dos resultados dos testes, destacando a importância de considerar eficiência computacional ao lidar com operações em imagens.

ANEXOS

Anexo 1 – *Função ImageLocateSubLocate()*

```
int ImageLocateSubImage(Image img1, int* px, int* py, Image img2) { ///
```

```
  assert (img1 != NULL);
  assert (img2 != NULL);
  // Insert your code here!
  int w1=img1->width;
  int h1=img1->height;
  int w2=img2->width;
  int h2=img2->height;



  // for each pixel in img1 check if it matches the first pixel of img2
  // if it does, check if the next pixel matches the next pixel of img2
  // if thats true for all pixels in img2, return 1 and the position of the first pixel
  // else, break and continue checking the next pixel in img1
  // if it cannot find a match, return 0
  //worst case? will look all over the image for the first pixel


  //best case? will look for first pixel where it must be for this to be a subpicture
  // meaning it wouldnt fit inside the Bigger image if isnt in that square so there is no
point looking elsewhere


   //for (int x1=0; x1<w1; x1++){ //worst case?
   for (int x1=0; x1<w1-w2; x1++){ //best case?
   //for (int y1=0; y1<h1; y1++){//worst case ?
     for (int y1=0; y1<h1-h2; y1++){ // bestcase?


       if (ImageMatchSubImage(img1,x1,y1,img2)){


         *px=x1;
         *py=y1;
         return 1;
       }
     }
```

```
    }
  return 0;
}
```

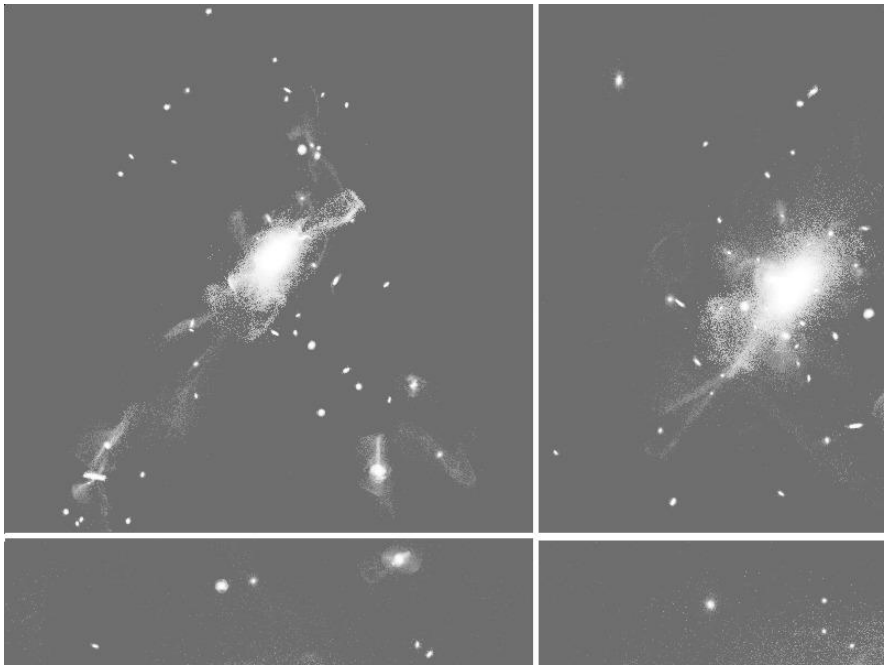
Anexo 2 - *Função ImageBlur()*

```
void ImageBlur(Image img, int dx, int dy) { /// Bruno
    assert (img != NULL);
```

```
assert (dx >= 0 && dy >= 0);

// Create a temporary image to store the blurred result
Image image2 = ImageCreate(img->width, img->height, img->maxval);

for (int i = 0; i < img->height; i++) {
    for (int j = 0; j < img->width; j++) {
        int sum = 0;
        int count = 0;
        for (int k = -dx; k <= dx; k++) {
            for (int l = -dy; l <= dy; l++) {
                int x = j + k;
                int y = i + l;
                if (x >= 0 && x < img->width && y >= 0 && y < img->height) {
                    sum += ImageGetPixel(img, x, y);
                    count++;
                }
            }
        }
        // Use round for proper rounding of the mean
        int mean;
        if (count > 0) {
            mean = (int)((float)sum / count + 0.5);
        } else {
            mean = 0;
        }
        // Convert the mean back to uint8
        uint8 meanInt = (uint8)mean;
        ImageSetPixel(image2, j, i, meanInt);
    }
}
```

```
    // Copy the blurred result back to the original image
    for (int i = 0; i < img->height; i++) {
        for (int j = 0; j < img->width; j++) {
            uint8 pixelValue = ImageGetPixel(image2, j, i);
            ImageSetPixel(img, j, i, pixelValue);
        }
    }

    // Destroy the temporary image
    ImageDestroy(&image2);
}
```

Anexo 3 – ***Imagem 800x600***



Anexo 4 – ***Imagem 300x300***