Relatório de AED

Departamento de Eletrónica, Telecomunicações e Informática

Martim Peralta Gomes, nº 119488

Tiago Queirós Rocha, nº 120515

Aspetos Gerais

Este relatório descreve o desenvolvimento e análise do TAD imageBW, realizado no âmbito do 1º trabalho da disciplina de Algoritmos e Estruturas de Dados. O objetivo deste trabalho foi criar um módulo para manipulação de imagens binárias, ou seja, imagens compostas por pixels que podem assumir apenas dois valores: preto (1) ou branco (0).

O uso da técnica de Run-Length Encoding (RLE), é uma forma de compressão sem perdas, utilizada ao longo deste trabalho. Com ela, a imagem é representada de maneira mais compacta, aproveitando as sequências de pixels consecutivos com o mesmo valor, o que facilita tanto o armazenamento quanto a manipulação das imagens.

Durante o desenvolvimento do código, completamos as funções que permitem criar e manipular imagens binárias. Entre as funcionalidades principais estão:

- A criação de imagens com padrões simples (como imagens totalmente brancas ou pretas) e padrões mais complexos, como o tabuleiro de xadrez.
- Operações lógicas entre imagens, como AND, OR e XOR.
- Transformações geométricas, como espelhamento horizontal e vertical.
- Leitura e escrita de imagens no formato PBM.

Além disso, também realizamos testes para avaliar o uso de memória das imagens criadas, utilizando diferentes dimensões e padrões observando como impactam no espaço ocupado. Analisamos ainda, a

complexidade das operações realizadas, em particular a função ImageAND, para entender a sua eficiência e comportamento computacional.

Neste relatório vamos analisar desde a implementação das funções até a análise de desempenho das mesmas, tendo como objetivo entender melhor como o TAD imageBW pode ser usado de forma eficiente e como podemos otimizar o uso de memória e processamento ao trabalhar com imagens binárias.

Análise da função *ImageCreateChessboard()*

1. Dados Experimentais

Analisando o espaço de memória ocupado pelas imagens criadas pela função ImageCreateChessboard() em função do seu número de linhas, de colunas e do comprimento da aresta de cada quadrado do padrão de xadrez criado obtivemos as seguintes expressões:

- Número de Runs por Linha = $\frac{Width}{Square\ Edge}$
- Número de Runs Total = $\frac{Width}{Square\ Edge} \times Height$
- Memória Ocupada por Linha = (Número de Runs por Linha + 2) × sizeof(uint32)
- Memória Total Ocupada = Memória Ocupada por Linha × $Height$ + 16 (header aloca memória para a altura e largura (4 + 4 + 8))

2. Análise do espaço de memória ocupado pela imagem criada

Com isto, foram realizados vários testes, de forma a verificar o total de runs, assim como a memória usada.

Teste	Width	Height	Square Edge	Total Runs	Memory Used (bytes) + 16
1	1	1	1	1	12
2	10	10	5	20	160
3	8	8	1	64	320
4	8	8	2	32	192
5	8	16	1	128	640
6	8	16	2	64	384
7	16	8	1	128	576
8	16	8	2	64	320
9	16	8	4	32	192
10	16	16	2	128	640
11	32	16	4	128	640
12	32	32	4	256	1280
13	64	64	8	512	2560
14	128	64	8	1024	4608
15	128	128	16	1024	5120
16	1024	1024	16	65536	270336

Figura 1 - Testes

Analisando os resultados, o padrão de xadrez que dá origem à imagem com menores runs, é quando todas as características do padrão são iguais a 1, totalizando um número de runs, também, igual a 1, com 12 bytes de memória utilizada.

Por outro lado, o padrão de xadrez que dá origem à imagem com menores runs, está dependente das suas características. Ou seja, quanto maior a altura e a largura da imagem e menor for o square edge, maior será o número de runs.

Análise da função *ImageAnd()*

1. Dados Experimentais

Na tabela seguinte encontram-se os resultados dos testes computacionais com imagens de diferentes tamanhos:

Algoritmo Otimizado		
Image Size	Nº Operations	Time
2	4	0.000003
4	8	0.000006
8	16	0.000005
16	32	0.000005
32	64	0.000007
64	128	0.000037
128	256	0.000150
256	512	0.000419
512	1024	0.000829
1024	2048	0.003070
2048	4096	0.006086
4096	8192	0.016990
8192	16384	0.032945
16384	32678	0.073754

Figura 2- Dados do Algoritmo Otimizado

2. Análise Formal

Na função base (não otimizada), fazemos sempre a descompressão da imagem e percorremos todos os pixéis de cada linha, logo, o melhor e o pior caso vão ser iguais. Sendo assim, podemos ver que a complexidade desta função é $O(n^2)$, caso os valores que atribuirmos ao n e ao m sejam iguais (o que é suposto, por ser um quadrado de xadrez), senão será O(n x m).

Pela observação dos resultados, podemos confirmar isso, fazendo a divisão do último tempo pelo penúltimo (0.771869/0.196200 ≃4), consultando a tabela dada nas aulas teóricas na aula 2, podemos verificar que este valor corresponde a uma função exponencial de complexidade $O(n^2)$.

Por outro lado, na **função otimizada**, já temos uma variação do melhor para o pior caso, onde o melhor caso será apenas quando fazemos um run por linha, ou seja, quando a linha é toda da mesma cor, obtendo assim uma complexidade O(n). Com isto, podemos concluir, novamente, pelos dados e pela divisão do último pelo penúltimo tempo (0.073754 / 0.032945 ≃ 2), que se trata de uma função linear, ou seja O(n).

Já no pior caso compara-se pixel a pixel tal como na função não otimizada , logo concluímos que o pior caso será $O(n^2)$.

3. Análise Comparativa: Algoritmo Básico/Algoritmo Melhorado

Aqui encontram-se os dados referentes à comparação entre os dois tipos de algoritmos:

Algoritmo Base	Image Size	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
	Time	0.000003	0.000002	0.000002	0.000004	0.000010	0.000037	0.000106	0.000384	0.000862	0.003170	0.012391	0.049429	0.196200	0.771869

Figura 3 - Dados Algoritmo Base

Algoritmo Otimizado	Image Size	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
	Time	0.000003	0.000006	0.000005	0.000005	0.000007	0.000037	0.000150	0.000419	0.000829	0.003070	0.006086	0.016990	0.032945	0.073754

Figura 4 - Dados Do Algoritmo Otimizado

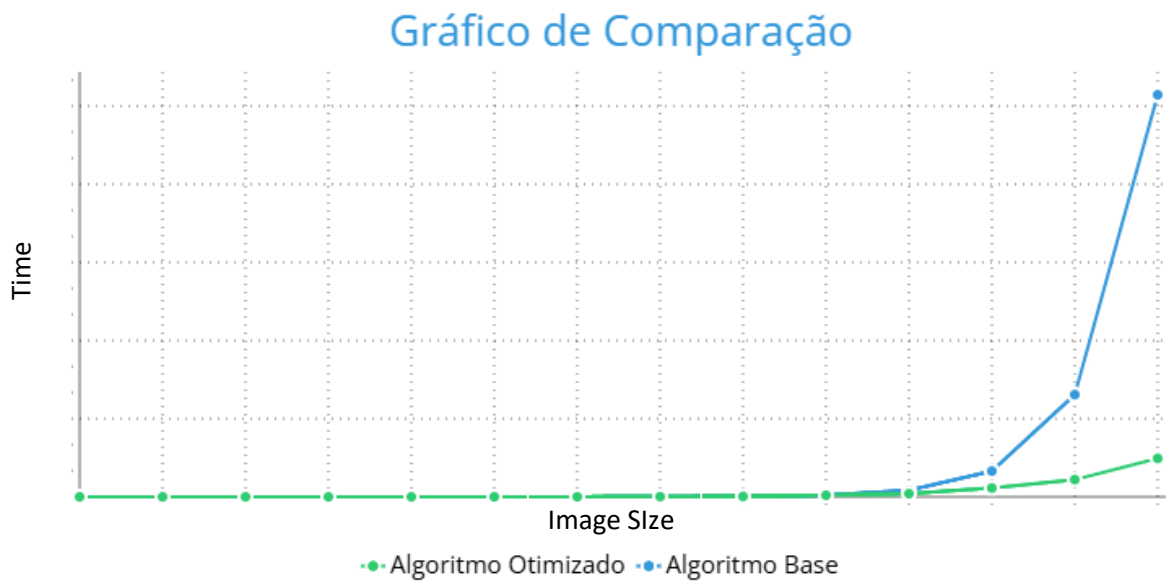

Figura 5- Gráfico de Comparação entre Algoritmos

Tendo em conta a análise do gráfico, podemos confirmar tudo o que já foi dito anteriormente. O mesmo reflete como o algoritmo otimizado, é mais viável e mais rápido em comparação com o algoritmo base, ao nível do tempo, aquando lida com imagens maiores, visto a comparação não ser feita pixel a pixel.

Dado que não é necessário fazer a compressão e descompressão das imagens, não sendo assim preciso armazenar as linhas descomprimidas para realizar a operação And, a função necessita de menos memória para realizar operações, sendo por isso o mais adequado e mais eficiente no que toca a imagens grandes ou com muitos pixéis.

Conclusão

Após finalização deste projeto, retiramos uma melhor perceção acerca da técnica usada (Run-Length Encoding), bastante eficaz na redução do espaço de armazenamento das imagens, especialmente quando se tratam de padrões simples e repetitivos.

Para além disso, através do estudo da complexidade de funções, tal como a ImageAND, permitiu-nos entender o impacto dessas operações em termos de tempo de execução, algo crucial para otimizar o desempenho em tarefas com imagens, operações lógicas, como também de transformações geométricas.

Ficamos a perceber, também, que o consumo de memória está diretamente ligado aos mais diversificados aspetos, como as características da imagem e o tamanho dos quadrados no padrão de xadrez. Imagens mais simples, apresentaram uma compressão mais eficiente, enquanto imagens com maior frequência de alternância ocupam mais memória derivado do aumento do número de runs.

De forma geral, o TAD imageBW mostrou-se uma ferramenta robusta e eficaz para manipulação de imagens binárias. Contudo, há possibilidades de aprimoramento, como a aplicação de algoritmos, possivelmente, ainda mais eficientes para cenários com alta alternância de forma a maximizar o desempenho.