AED trabalho 1 - O TAD imageBW

Xavier Baltarejo nº118832 João Barreira nº120054

30 novembro 2024

---

1 Introdução

No contexto da disciplina de Algoritmos e Estruturas de Dados (AED), este projeto foca-se no desenvolvimento e análise de um Tipo Abstrato de Dados (TAD) denominado `imageBW`. Este TAD permite a manipulação de imagens binárias (BW - *Black and white*). Estas são representadas em memória de forma compacta, usando *Run Length Encoding* (RLE), para reduzir o consumo de memória e melhorar a eficiência de processamento. Cada linha comprimida começa com o valor da cor do primeiro pixel, seguido pelos comprimentos das sequências consecutivas da mesma cor, terminando com o marcador especial EOR (do inglês *End Of Row*) de valor `-1`, que indica o fim da linha.

Um dos objetivos deste trabalho foi implementar e testar operações em imagens binárias, otimizando o uso de memória e o desempenho computacional. As operações implementadas incluem operações lógicas (`AND`, `OR`, `XOR`) e manipulações geométricas, como espelho e replicação, tanto horizontais como verticais.

Para além do desenvolvimento do TAD `imageBW`, o projeto envolveu a análise do uso de memória para imagens geradas pela função `ImageCreateChessboard`. Também realizamos uma análise completa ao desempenho e complexidade da função `ImageAND`, comparando 2 estratégias algorítmicas distintas.

2 Análise ImageCreateChessboard

Desenvolvemos a função `ImageCreateChessboard`, que recebe como argumentos uma largura **m**, uma altura **n**, um lado para os quadrados interiores **s** e também a cor do primeiro quadrado **c**, por esta ordem. Esta retorna uma nova imagem que representa um tabuleiro de xadrez. Segue-se um exemplo:

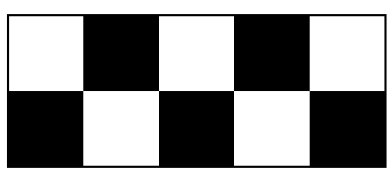

Figure 1: resultado imageCreateChessboard(5,2,1,0)

Para o estudo do espaço conforme a mudança no número de linhas **m**, colunas **n** ou lado **s**, consideramos três experiências diferentes onde para cada uma se faz variar uma das 3 variáveis. O estudo do tamanho da imagem não consideranda o tamanho do *header* da estrutura, uma vez que este é constante e é desprezível para imagens grandes.

1) Variar m: Ao variar **m** percebe se que o número de *runs* aumenta linearmente com o aumento do número de colunas, assim como o tamanho ocupado.

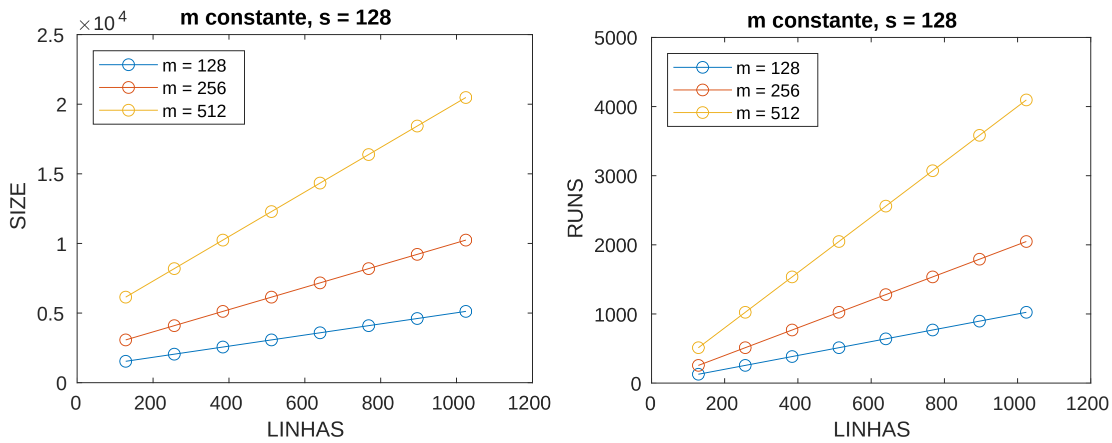

Figure 2: resultados experimentais

2) Variar n: Neste caso, as relações verificadas anteriormente mantém-se validas, sendo ambas lineares.

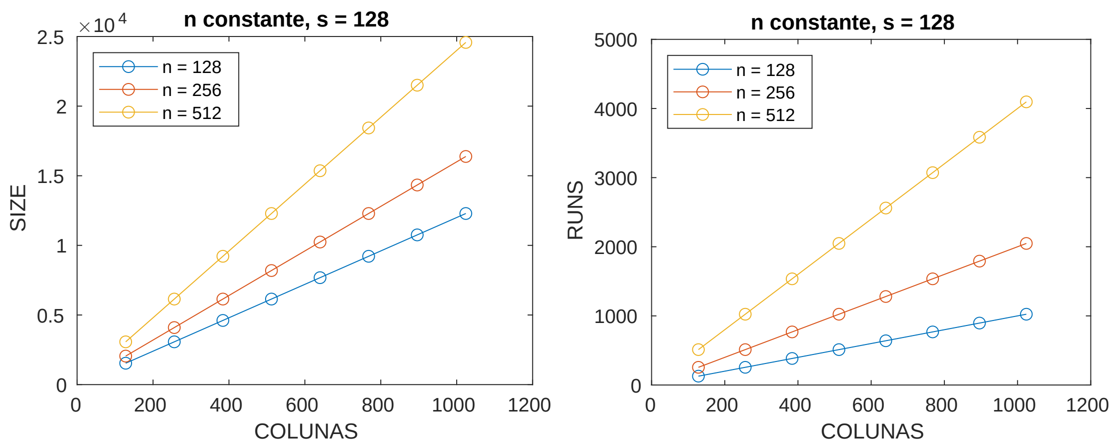

Figure 3: resultados experimentais

3) Variar s: Fazendo variar o lado dos quadrados interiores, obtemos este gráfico que se assemelha a um ramo de uma hipérbole, indicando uma relação inversa com o número de *runs* e tamanho da imagem.

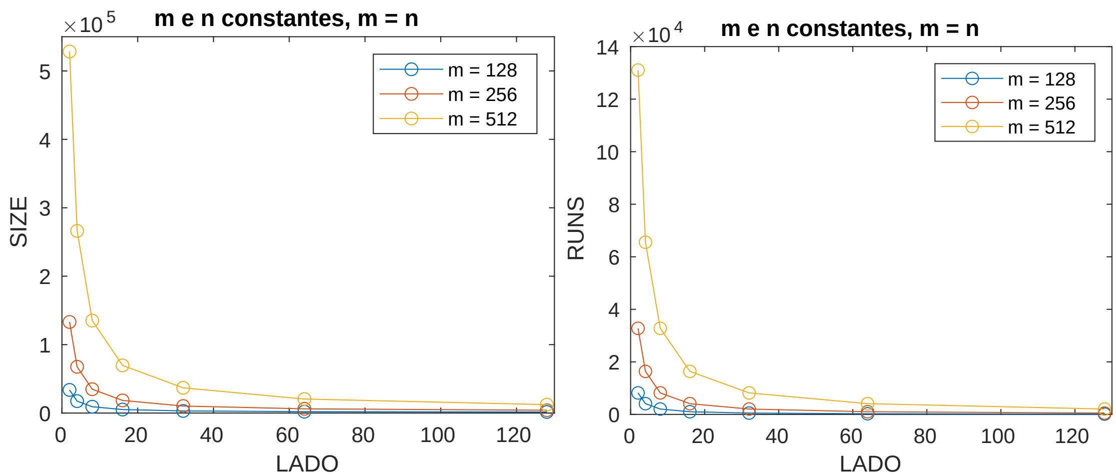

Figure 4: resultados experimentais

Depois de analisados os dados, e sabendo que o número de *runs* é igual para todas as linhas, deduzimos o seguinte:

$$runsporlinha = n/s$$

$$runs = n/s * m$$

$$size = (runsporlinha + 2) * m * 4$$

Onde *size* é o tamanho em *bytes* da imagem, não incluindo o *header* da estrutura. Dito isto, concluímos que a imagem com maior número de *runs* é aquela cujo **s** é mínimo, ou seja 1, o que equivale a $n \times m$ *runs* no total. O caso que minimiza o número de *runs* é aquela cujo **s** é máximo, ou seja m, o que corresponde a n *runs*.

Estas formulas concordam com os dados experimentais, por isso sentimos-nos seguros dos nossos resultados.

3 Análise ImageAND

As funções `ImageAND` e `ImageAND2` implementam a operação lógica AND entre duas imagens, de iguais dimensões, utilizando abordagens distintas que afetam diretamente o desempenho computacional. Ambas recebem como argumentos duas imagens, e retornam uma nova imagem.

- `ImageAND`:
  - Opera ao nível dos pixeis das imagens, realizando a operação lógica pixel a pixel.
  - Utiliza métodos de compressão e descompressão para obter os valores dos pixeis.
- `ImageAND2`:
  - Trabalha diretamente no formato RLE, evitando a descompressão e compressão.

Para o estudo da complexidade dos algoritmos foi considerada como operação de maior custo computacional o acesso à memória onde está guardada a informação dos pixeis, tanto das linhas comprimidas como descomprimidas, logo toda a nossa análise foca-se na contagem destas operações. Estabelecemos 3 casos de estudo: o pior caso, melhor caso e um caso médio. Como imagens de teste usamos aquelas criadas pela `ImageCreateChessboard`, com $s = 1$ para o pior caso, maximizando o número de *runs*, e com $s = 2$ para o caso médio.

Considerando a experiência aleatória: "Gerar dois vetores de inteiros aleatórios de 1 a n de tamanho n", que representam as *runs* para as n linhas de uma imagem. A soma dos dois vetores para uma amostra de 10^6 experiências resultou numa distribuição normal de média $\approx n$ que equivale a $n/2$ para cada linha. Isto justifica o uso das imagens geradas por `ImageCreateChessboard` com $s = 2$, que gera imagens com $w/2$ *runs* por linha. (Ver código `test/proof.m`).

Já no melhor caso usamos a função `ImageCreate` para obter uma imagem totalmente branca/preta. Todas as imagens de teste são quadradas de dimensões: $size \times size$. Seguem-se exemplos de imagens usadas para cada um dos testes:

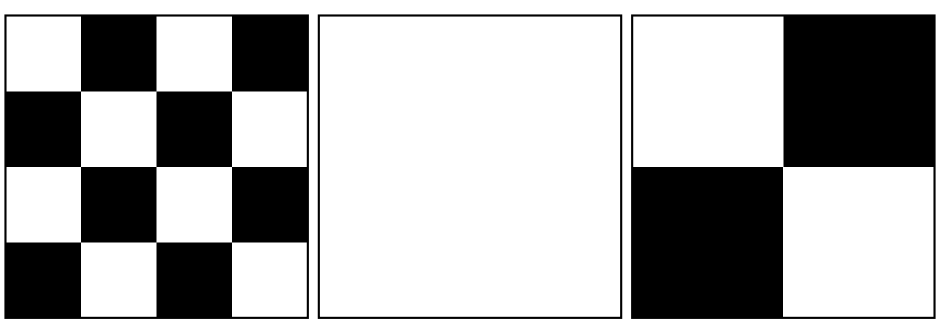

Figure 5: Exemplos de imagens usadas no pior, melhor e caso médio respetivamente

3.1 Descompressão/Compressão (ImageAND)

Etapas do Algoritmo:

1. **Descompressão:** Cada linha é descompactada para o formato RAW.
2. **Operação Pixel a Pixel:** A operação lógica AND é realizada para cada pixel.
3. **Compressão:** Cada linha resultante é compactada novamente para o formato RLE.

Analise formal: $W = largura\ H = altura$
A expressão $\sum_{h=0}^{H}$ deve-se à iteração por cada linha das imagens. Tanto em C como em D os somatórios foram traduzidos dos ciclos implementados, isto aplica-se à figura 6 e 7. $\sum_{w=1}^{W} 3$ diz respeito ao ciclo onde é feita a operação lógica pixel a pixel.

Nota: C refere-se à complexidade do algoritmo de compressão, e D à de des -compressão. Estas também variam conforme a imagem.

$$C = 3 + \sum_{w=1}^{W-1} 2 = 3 + 2(W-1) = 2W + 1 \qquad D = 3 + \sum_{w=0}^{W-1} 2 = 2W + 3$$

$$\sum_{h=0}^{H-1}(2D + \sum_{w=1}^{W} 3 + C) = \sum_{h=0}^{H-1}(2(2W+3) + 3W + (2W+1)) = \sum_{h=0}^{H-1}(9W+7) = (9W+7) \times H$$

Figure 6: Complexidade do melhor caso

$$C = 3 + \sum_{w=1}^{W-1} 3 = 3 + 3(W-1) = 3W \qquad D = 2 + \sum_{w=0}^{W-1} 3 = 3W + 2$$

$$\sum_{h=0}^{H-1}(2D + \sum_{w=1}^{W} 3 + C) = \sum_{h=0}^{H-1}(2(3W+2) + 3W + 3W) = \sum_{h=0}^{H-1}(12W+4) = (12W+4) \times H$$

Figure 7: Complexidade do pior caso

Para o número de operações AND este é sempre $W \times H$ uma vez que a operação é feita pixel a pixel, e existem $W \times H$ pixeis.

3.2 Algoritmo RLE (ImageAND2)

A ideia por detrás deste algoritmo surgiu ao entendermos o enorme custo computacional que a compressão e descompressão acarretam. Pensando num cenário em que a imagem é guardada sem compressão, a complexidade do primeiro algoritmo seria de $3W \times H$ uma vez que eram precisos 3 acessos à memoria (2 para consulta e 1 para atribuição na nova imagem). Como já estudamos anteriormente a complexidade triplica e até quadruplica no pior caso. Para evitar isto, acedemos diretamente ao valor das *runs*.

Etapas do Algoritmo:

1. **Determinar cor inicial:** É determinado o valor do primeiro pixel.
2. **Calculo das *Runs*:** É feito o calculo do tamanho das *runs* consequentes, percorrendo iterativamente por ambas as linhas comprimidas das imagens de entrada.

Análise formal: W = largura, H = altura
A expressão $\sum_{h=0}^{H}$ deve-se à iteração por cada linha das imagens. A constante 6 refere-se aos acessos que acontecem antes e depois do acesso aos valores das *runs*. Na figura 8, $\sum_{w=1}^{W-1} 6$ deve-se à iteração pelos valores das *runs*. Já na figura 9, a constante 2 aparece uma vez que só existem 2 *runs* e só são feitas 2 leituras.

$$\sum_{h=0}^{H-1}(6+\sum_{w=1}^{W-1}6) = \sum_{h=0}^{H-1}(6+6(W-1)) = \sum_{h=0}^{H-1}(6W) = 6WH$$

Figure 8: Complexidade do pior caso

$$\sum_{h=0}^{H-1}(6+2) = \sum_{h=0}^{H-1}(8) = 8H$$

Figure 9: Complexidade do melhor caso

O número de operações AND nos pixeis é sempre 1 por linha o que equivale a H operações por imagem. Esta operação está associada a determinar o valor do pixel inicial.
Devido á sua complexidade não realizamos a analise formal do caso médio.

3.3 Avaliação dos resultados experimentais

No cenário do pior caso como existe um grande número de *runs*, são necessários mais acessos à memória. Apesar de a função `ImageAND2` ter uma grande vantagem, um elevado número de *runs* prejudica o seu desempenho.

Nota: `PIXMEM` refere-se ao número de acessos à memoria onde os pixeis, ou as *runs*, se encontram

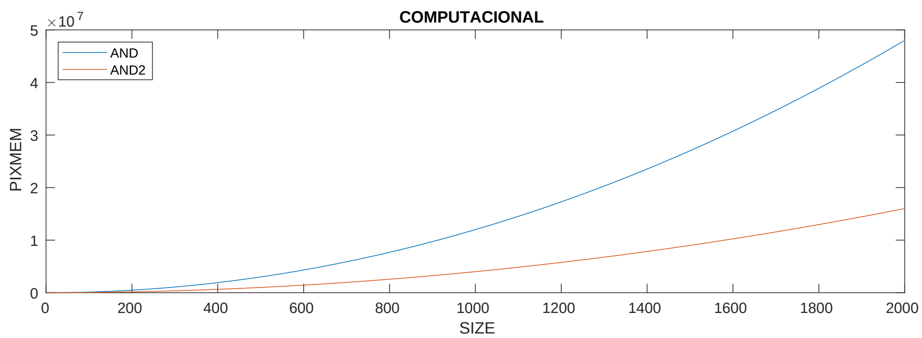

Figure 10: Pior caso, compressão mínima ($s = 1$).

Já no caso médio, a função `ImageAND2` apresenta ganhos de eficiência ainda maiores em relação à `ImageAND`, aproveitando a compressão parcial.

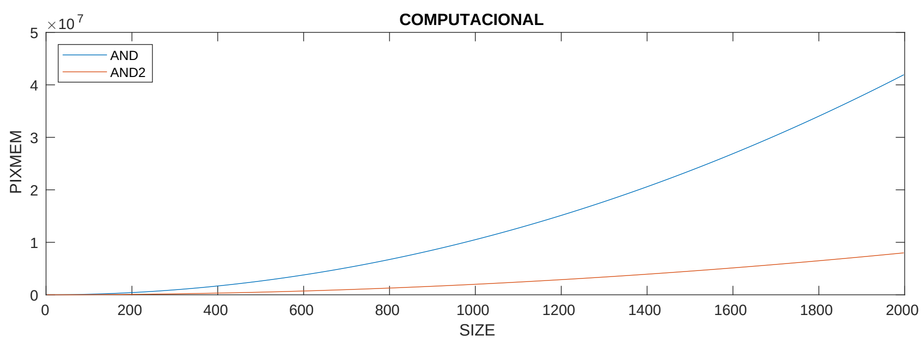

Figure 11: Caso médio, compressão intermediária ($s = 2$).

A maior diferença é mesmo no melhor caso em que a função `ImageAND2` é extremamente eficiente, com um número bastante baixo de acessos à memoria, enquanto a `ImageAND` mantém um elevado número de acesso à memoria devido à necessidade de descompressão e compressão.

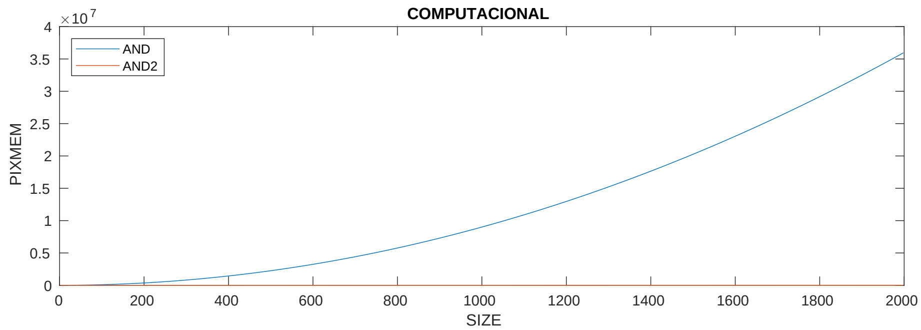

Figure 12: Melhor caso, compressão máxima.

Nota: na Figura 12 os valores de `ImageAND2` parecem nulos, mas na realidade seguem a expressão deduzida na Figura 9.

Feita a análise, estamos confiantes nos nossos resultados, tanto teóricos como experimentais, uma vez que se encontram em concordância. Também ficamos bastante satisfeitos com os ganhos computacionais que conseguimos usando o algoritmo otimizado, que tem melhor eficiência quanto mais comprimida a imagem estiver, batendo ainda o algoritmo inicial na situação de menor compressão.

4 Conclusão

No âmbito deste projeto, desenvolvemos as funções especificadas no ficheiro `imageBW.c` e garantimos o seu correto funcionamento. Desenvolvemos ainda a função `ImageAND2` ao entendermos o enorme custo computacional que a compressão e descompressão têm.

Com base na análise realizada sobre a função `ImageCreateChessboard`, concluímos que o número de *runs* e o espaço ocupado pela imagem dependem diretamente dos parâmetros m, n e s. Observou-se que, ao variar m ou n, o número de *runs* cresce linearmente, evidenciando a relação proporcional entre as dimensões da imagem e a quantidade de dados armazenados. Por outro lado, ao variar s, o comportamento muda significativamente: o número de *runs* apresenta uma relação inversa com s, com o máximo de *runs* em $s = 1$ e o mínimo em $s = m$.

Para as funções `ImageAND` e `ImageAND2`, foram analisados os casos possíveis e o respetivo número de operações de acessos à memoria do pixeis/*runs*, o que nos permitiu entender as vantagens de trabalhar diretamente no formato comprimido.

Este trabalho representou um desafio enriquecedor, combinando conceitos de compressão de dados, análise de complexidade e otimização algorítmica. Além disso, permitiu-nos explorar diferentes estratégias de desenvolvimento e validação, utilizando ferramentas como o `imageTool` e `instrumentation.h`, o que proporcionou uma análise mais detalhada e robusta do desempenho do TAD imageBW. Os resultados obtidos e as conclusões apresentadas constituem uma base sólida para futuras otimizações e aplicações em manipulação de imagens comprimidas.