Relatório do Trabalho 1
Algoritmos e Estruturas de Dados (2024/2025)

Afonso Correia Sampaio (119751)
Isabela de Matos Pereira (119931)

4 de dezembro de 2024

Introdução

No contexto da disciplina de Algoritmos e Estruturas de Dados, este projeto foca-se no desenvolvimento e análise do Tipo Abstrato de Dados (TAD) imageBW. Este TAD é responsável por representar e manipular imagens binárias (preto e branco), onde cada pixel pode assumir apenas dois valores de intensidade: 0 (branco) ou 1 (preto).
Para garantir uma representação eficiente, o TAD utiliza uma técnica de compressão sem perdas chamada Run-Length Encoding (RLE). Essa técnica compacta sequências consecutivas de pixels com o mesmo valor, reduzindo o espaço de armazenamento necessário.

O trabalho envolve três principais desafios:

- **Desenvolvimento do TAD imageBW:** Implementação e teste de funções específicas definidas na interface imageBW.h.
- **Análise do Espaço de Memória:** Avaliação do uso de memória da função ImageCreateChessboard() em função dos parâmetros da imagem e da compressão RLE.
- **Análise da Complexidade:** Estudo da eficiência computacional da função ImageAND(), considerando a quantidade de operações necessárias e a comparação entre abordagens algorítmicas.

Para testar e validar as funções desenvolvidas, foram criados os ficheiros `ImageAnd_Tests.c` e `chessboard_ras.c` que estruturam os dados para poderem ser importados para um ficheiro *csv*. A metodologia de teste é baseada na execução desses programas em C da função main `imageBW.c`, cujos resultados são processados e apresentados graficamente através do *Excel* para facilitar a análise.

Análise da Função `ImageCreateChessboard()`

0.1 Análise Formal

A função cria imagens binárias com padrão de xadrez, onde cada quadrado possui arestas de comprimento s, utilizando codificação por comprimentos de execução, conhecida como Run-Length Encoding (RLE).

O melhor caso é $O(R)$ e ocorre quando a imagem possui o menor número possível de R, ou seja, quando o s = n, onde R é igual a m:

$$Bestcase = \Omega(R)$$

, onde

$$R = m$$

O pior caso é $O(m * n)$ e ocorre quando cada pixel é um *run* individual ou seja a imagem alterna de cor a cada píxel, tanto na horizontal como na vertical

0.2 Resultados Experimentais

Criaram-se imagens com diferentes dimensões e tamanhos de quadrados. A Figura 1 apresenta a memória ocupada e os *runs* em função dos parâmetros m, n e s.

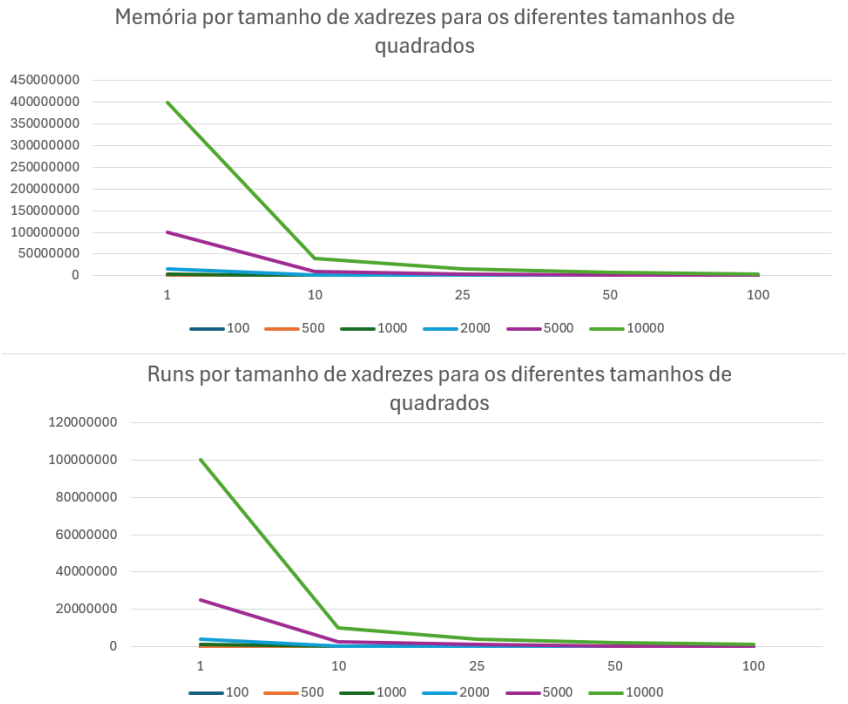

Figura 1: Memória ocupada e os *runs* por padrões de xadrez com diferentes valores de s. Tabela com os valores experimentais no anexo 7 e no anexo 8

Destes Gráficos conseguimos concluir as seguintes informações:

- **Caso de Menor Uso de Memória e de *runs*:**

  O menor número de *runs* ocorre quando s é igual a n (o quadrado ocupa toda a largura da imagem).

  Neste caso, há apenas 1 *run* por linha e o número total de *runs* vai apenas depender de m.

- **Caso de Maior Uso de Memória e de *runs*:**

  O maior número de *runs* ocorre quando $s = 1$ (cada pixel é um quadrado de tamanho 1).

  Há n *runs* por linha, resultando em máximo uso de memória visto que $R = m \times n$.

- **Efeito do Tamanho do Quadrado (s):**

  À medida que s aumenta, o número de quadrados por linha diminui:

$$\text{Número de quadrados por linha} = \frac{n}{s}$$

Isso resulta em menos *runs* por linha e, consequentemente, menos elementos RLE e menor uso de memória.

0.3 Expressão Analítica

O número total de runs na imagem é dado por:

$$\boxed{R = m \times \frac{n}{s}}$$

- Em cada linha da imagem, a cor alterna a cada s pixel, formando quadrados de tamanho $s \times s$.
- O número de *runs* por linha é, portanto, $\frac{n}{s}$.
- Multiplicando pelo número total de linhas m, obtemos o número total de *runs* na imagem.

A Memória total é dada por:

$$\boxed{\text{Memória\_total} = m \times \left[\left(\frac{n}{s} + 2\right) \times \text{sizeof(int)} + \text{sizeof(int*)}\right] + \text{sizeof(ImageStruct)}}$$

Explicação:

- **Por linha:**
  - Cada linha RLE possui:
    - Um valor inicial do pixel (1 inteiro).
    - Um conjunto de comprimentos de *runs* (um inteiro para cada *run*).
    - Um marcador de fim de linha (EOR) (1 inteiro).
  - Total de inteiros por linha: $\left(\frac{n}{s} + 2\right)$.
  - Memória ocupada pelos inteiros: $\left(\frac{n}{s} + 2\right) \times \text{sizeof(int)}$.
- Além disso, há um ponteiro para o array RLE da linha: sizeof(int*).
- **Total na imagem:**
  - Multiplicando pela quantidade de linhas m, temos a memória ocupada por todas as linhas.
  - Somando o tamanho da estrutura de cabeçalho da imagem sizeof(ImageStruct), obtemos a memória total.

Definição dos Parâmetros As expressões são em função de:

- n: largura da imagem (número de pixels por linha).
- m: altura da imagem (número de linhas).
- s: tamanho da aresta de cada quadrado (número de pixels).
- sizeof(int): tamanho de um inteiro em bytes (tipicamente 4 bytes).
- sizeof(int*): tamanho de um ponteiro para inteiro em bytes (tipicamente 8 bytes em sistemas de 64 bits e 4 em sistemas de 32 bits).
- sizeof(ImageStruct): tamanho da estrutura de cabeçalho da imagem(tipicamente 16 bytes em sistemas de 64 bits e 12 em sistemas de 32 bits).

Análise da Função `ImageAND()`

Esta função combina duas imagens binárias, aplicando a operação lógica AND pixel a pixel. A complexidade computacional foi analisada formalmente e validada por testes experimentais.

0.4 Análise Formal

A função `ImageAND()` contém dois loops, um loop externo e outro interno. O loop externo percorre todas as linhas (*HEIGHT*), enquanto que o loop interno percorre os pixéis de cada linha (*WIDTH*).
A complexidade é, então, linear em relação ao número de pixéis:

$$O(w \cdot h)$$

Reforçamos também que a complexidade do algoritmo vai depender diretamente *SIZE = (w * h)* e não da imagem em si, por exemplo, a complexidade do algoritmo em questão de uma imagem xadrez é a mesma que a de uma imagem toda a preto ou toda a branco. A prova experimental encontra-se em anexo 9.

0.5 Resultados Experimentais

0.5.1 Complexidade de `ImageAND()`

Um teste experimental foi realizado à função `ImageAND()` para comprovar a ordem de complexidade analisada anteriormente. Para tal, foi criada uma função para testes, a `ImageAnd_Tests.c`, onde se criou um array com seis larguras de imagem (*WIDTH*) diferentes e um inteiro para a altura da imagem que seria variado comforme a necessidade , o *HEIGHT*. Os dados foram depois exportados para um ficheiro *csv* e posteriormente importados para *Excel*, onde foram processados.

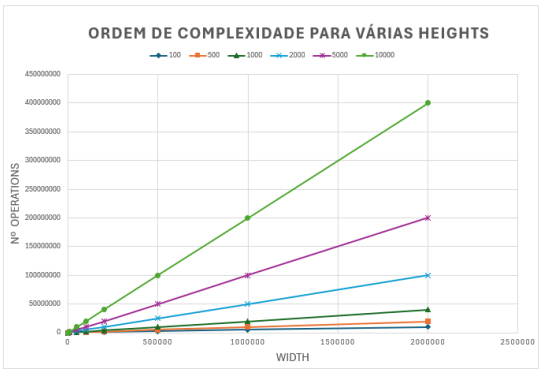

Figura 2: Número de operações pela *WIDTH* da imagem para várias *HEIGHT*. Tabela com os valores experimentais no anexo 10

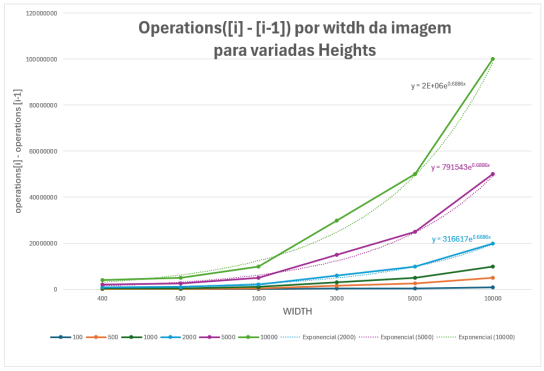

Figura 3: Taxa de variação das operações para várias larguras (*WIDTH*), entre diferentes alturas (*HEIGHT*)

O gráfico 2 apresenta para várias *HEIGHTS* o número de operações realizadas pela largura da imagem *WIDTH*. Podemos observar que, para larguras pequenas, o número de operações cresce de forma aproximadamente linear. À medida que a largura aumenta, especialmente para alturas maiores, o crescimento torna-se mais acentuado, sugerindo uma dependência proporcional tanto da largura quanto da altura.

O gráfico 3, analisa as diferenças incrementais entre operações para pares consecutivos de alturas, ou seja, compara `operations[i] − operations[i−1]`. O comportamento exponencial observado mostra que, conforme a largura aumenta, o impacto da altura torna-se mais pronunciado, o que evidência uma relação multiplicativa entre as duas variáveis h (altura) e w (largura).

Comfirma-se, então, pela análise dos dois gráficos, que a complexidade aumenta linearmente com o aumento do tamanho da imagem, pelo que a sua ordem de complexidade é:

$$O(w \cdot h)$$

, e sendo, um algoritmo determinista.

0.5.2 Comparação Algoritmo Básico com o Melhorado

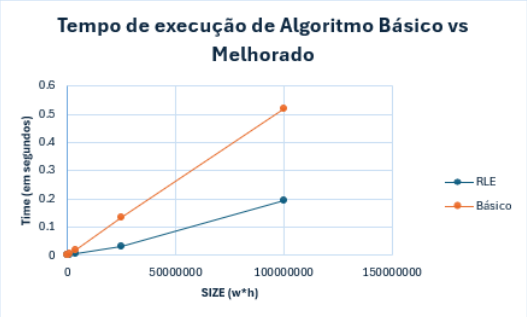

Figura 4: Tempo de execução por *SIZE* do algoritmo básico comparado com o algoritmo melhorado com RLE. Tabela com os valores experimentais no anexo 11.

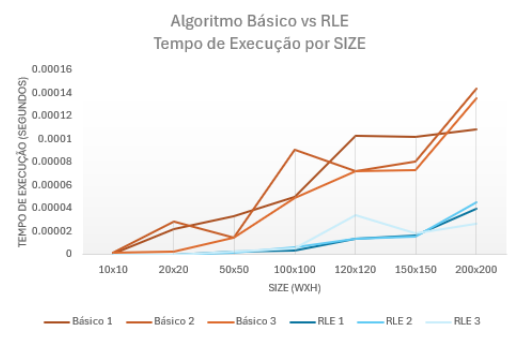

Figura 5: Tempo de execução baseados em 3 triagens de dados para SIZES menores do algorimo básico (a tons de vermelho) vs o algoritmo melhorado (a tons azuis).

O gráfico 4 apresenta o tempo de execução total dos dois algoritmos para tamanhos de entrada maiores. O algoritmo básico apresenta um crescimento mais acentuado no tempo de execução conforme o tamanho aumenta, evidenciando uma complexidade proporcional ao tamanho da imagem (wxh). Já o algoritmo RLE demonstra um crescimento mais contido, indicando uma otimização significativa, especialmente em tamanhos maiores. Essa diferença reflete a vantagem do RLE, que comprime a imagem antes do processamento, reduzindo eficazmente o número de operações.

No gráfico 5 observa-se que o tempo de execução do RLE mantém-se significativamente menor em todos os tamanhos, especialmente nos maiores, onde a compressão traz mais benefícios. O Algoritmo Básico apresenta um aumento gradual e consistente no tempo de execução à medida que o tamanho aumenta, confirmando a relação linear entre o tempo e o número total de operações necessárias.

A análise comparativa entre o Algoritmo Básico e o Algoritmo RLE demonstra claramente a superioridade do RLE em termos de tempo de execução, especialmente para tamanhos maiores. Esta técnica de compressão permite uma redução significativa do número de operações, confirmando, assim, a eficácia do algoritmo melhorado.

Ao compararmos o número de operações do algoritmo básico com o melhorado, verificamos que o número de operações do algoritmo aplicado diretamente em RLE é sempre metade do número de operações do algoritmo básico, tal como demonstrado pela tabela 6, através do cálculo da razão.

SIZE	RLE	B	B/RLE
100x100	5000	10000	2
500x500	125000	250000	2
1000x1000	500000	1000000	2
5000x5000	12500000	25000000	2
10000x10000	50000000	100000000	2

Figura 6: Número de operações para diferentes SIZES (imagens quadradas) para o Algoritmo básico com o Melhorado.

$$N^{\circ} oper Básico = 2 \times N^{\circ} oper Melhorado$$

Conclusões

O trabalho demonstrou a eficiência do TAD `imageBW` e a viabilidade de compactação usando RLE. As análises realizadas destacaram os benefícios do algoritmo otimizado para `ImageAND()`, bem como a dependência dos padrões de xadrez nos consumos de memória.

Anexo

Tabelas `ImageCreateChessboard()`

	100	500	1000	2000	5000	10000
1	41616	1008016	4016016	1.6E+07	1E+08	4E+08
10	5616	108016	416016	1632016	1E+07	4E+07
25	3216	48016	176016	672016	4080016	1.6E+07
50	2416	28016	96016	352016	2080016	8160016
100	2016	18016	56016	192016	1080016	4160016

Figura 7: Memória por tamanho de xadrezes para os diferentes tamanhos de quadrados

	100	500	1000	2000	5000	10000
1	10000	250000	1000000	4000000	2.5E+07	1E+08
10	1000	25000	100000	400000	2500000	1E+07
25	400	10000	40000	160000	1000000	4000000
50	200	5000	20000	80000	500000	2000000
100	100	2500	10000	40000	250000	1000000

Figura 8: *Runs* por tamanho de xadrezes para os diferentes tamanhos de quadrados

Prova 1

Segue-se a prova experimental de que a complexidade do algoritmo `ImageAND()` só depende do seu SIZE e não do seu *pattern*.
Para três imagens com padrões distintos, as quais, um xadrez, uma branca e uma preta, foram feitas três comparações AND bitwise com cada uma delas.

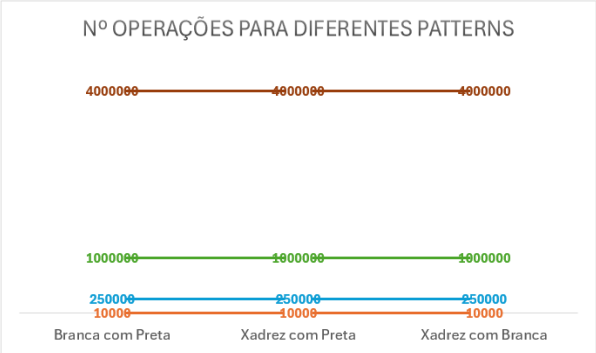

Figura 9: Número de operações para *patterns* distintos

Comprova-se, então, através da análise do gráfico 9 a veracidade da afirmação anterior, pois, o número de operações para os diferentes patterns manteve-se constante.

Tabelas `ImageAND()`

width \ height	100	500	1000	2000	5000	10000	20000
100	10000	50000	100000	200000	500000	1000000	2000000
500	50000	250000	500000	1000000	2500000	5000000	10000000
1000	100000	500000	1000000	2000000	5000000	10000000	20000000
2000	200000	1000000	2000000	4000000	10000000	20000000	40000000
5000	500000	2500000	5000000	10000000	25000000	50000000	100000000
10000	1000000	5000000	10000000	20000000	50000000	100000000	200000000
20000	2000000	10000000	20000000	40000000	100000000	200000000	400000000

Figura 10: Número de operações pelo SIZE

SIZE(w*h)	RLE	Básico
10000	0.000016	0.000074
250000	0.000322	0.001235
1000000	0.001541	0.005662
4000000	0.007945	0.020218
25000000	0.033081	0.134556
100000000	0.192706	0.520628

Figura 11: Tempo de execução por SIZE para O algoritmo Básico e Melhorado