Relatório AED – Projeto 1

TAD *imageBW*

Dinis Oliveira, 119193

André Silva, 119480

Turma P9

Considerações gerais

Este trabalho explora a manipulação de imagens comprimidas atravez do formato RLE, destacando duas funções principais: *ImageCreateChessboard* e *ImageAND*. A primeira função gera imagens com padrões de xadrez eficientes, já a segunda realiza operações and lógicas entre duas imagens comprimidas. Com o objetivo de analisar o impacto das diferentes formas de concretizar estas funções, e tendo por objetivo a sua otimização seja em termos de espaço ocupado, seja em termos de tempo, essas funções ilustram a aplicação prática de algoritmos eficientes.

Função ImageCreateChessboard()

Argumentos fornecidos e variáveis internas relevantes

- Width(n) - largura da imagem (pixéis);
- Height(m) - altura da imagem (pixéis);
- Square_edge(s) - comprimento de cada quadrado (pixéis)
- First_value(f) - primeiro valor do pixel (BLACK/WHITE)
- NsquaresX $\left(\frac{n}{s}\right)$- número de quadrados por linha;

Espaço Ocupado Pela Imagem

Cabeçalho da Imagem

Armazenamento dos dados raiz: width, height e o proprio array. Esta parte ocupa sempre um espaço constante, ou seja O_1. Sendo width e height int32 , o seu tamanho vai ser 4 logo 4+4 = 8 + 8(por causa do array) = 16.

Linhas Codificadas (RLE)

Cada linha (i) tem um *array* RLE para os NsquaresX quadrados da linha. Cada quadrado é representado por dois elementos, Valor do pixel (1/0) e comprimento da sequência(s). Ainda no final existe sempre o marcador EOR(End of Row).

Espaço por linhas

Para uma linha n:

- O número de quadrados NsquaresX$\left(\frac{n}{s}\right)$;
- Valores adicionais na sequência = 1valor do pixel + 1 valor final (EOR) = 2
- Sizeof(uint32) = 4

Logo o total de espaço por linha vai ser:

$$Espaço\ por\ linha\ = \left(\frac{n}{s} + 2\right) \times 4$$

Sendo assim, a equação que nos dá o espaço ocupado na memória é a seguinte:

$$E(m, n, s)\ = m \times \left(\ \left(\frac{n}{s} + 2\right) \times 4\right) + 16$$

De acordo com a equação, estes são os dados após alguns testes realizados:

WIDTH | HEIGHT | S_EDGE			TOTAL RUNS	MEMORY USED (bytes) + 16 (header)
4	**4**	**1**	16	112
4	**4**	**2**	8	80
4	**4**	**4**	4	64
8	**8**	**1**	64	336
8	**8**	**2**	32	208
8	**8**	**4**	16	144
8	**8**	**8**	8	1012
16	**16**	**4**	64	400
20	**20**	**5**	80	496
32	**32**	**1**	1024	4368
32	**32**	**4**	256	1296
32	**32**	**8**	128	784
128	**128**	**16**	1024	5136
256	**256**	**32**	2048	10256

Análise Formal de Complexidade

Tendo em conta a equação do espaço total e os dados da tabela, verificamos que o espaço cresce linearmente em relação ao número de linhas e quadrados por linha, mas é inversamente proporcional ao comprimento da aresta(s). Quanto maior o valor (s) menos espaço ocupa.

Melhor Caso (B(n))

- Condição: s = n, ou seja quanto maior o comprimento das arestas, menos espaço vai ser necessário para criar a imagem pedida.
- Comportamento: O loop interno realiza apenas uma iteração.
- $B(n) = 12 \times m$
- Complexidade: $T(n) = O(m)$

Pior Caso (W(n))

- Condição: s = 1, alternância entre pixéis, a cada linha (Linha RLE: [X,1,1,...,1]).
- Comportamento: O loop interno vai correr n vezes.
- $W(n) = 4 \times m(n+2)$
- Complexidade: $T(n) = (m \times n)$

Função ImageAnd()

A função realiza a operação lógica **AND** entre duas imagens de mesmas dimensões, com duas versões: uma que usa compressão/descompressão em RLE e outra que opera diretamente no formato comprimido, retornando a imagem resultante.

Descrição do Algoritmo

Estrutura geral:

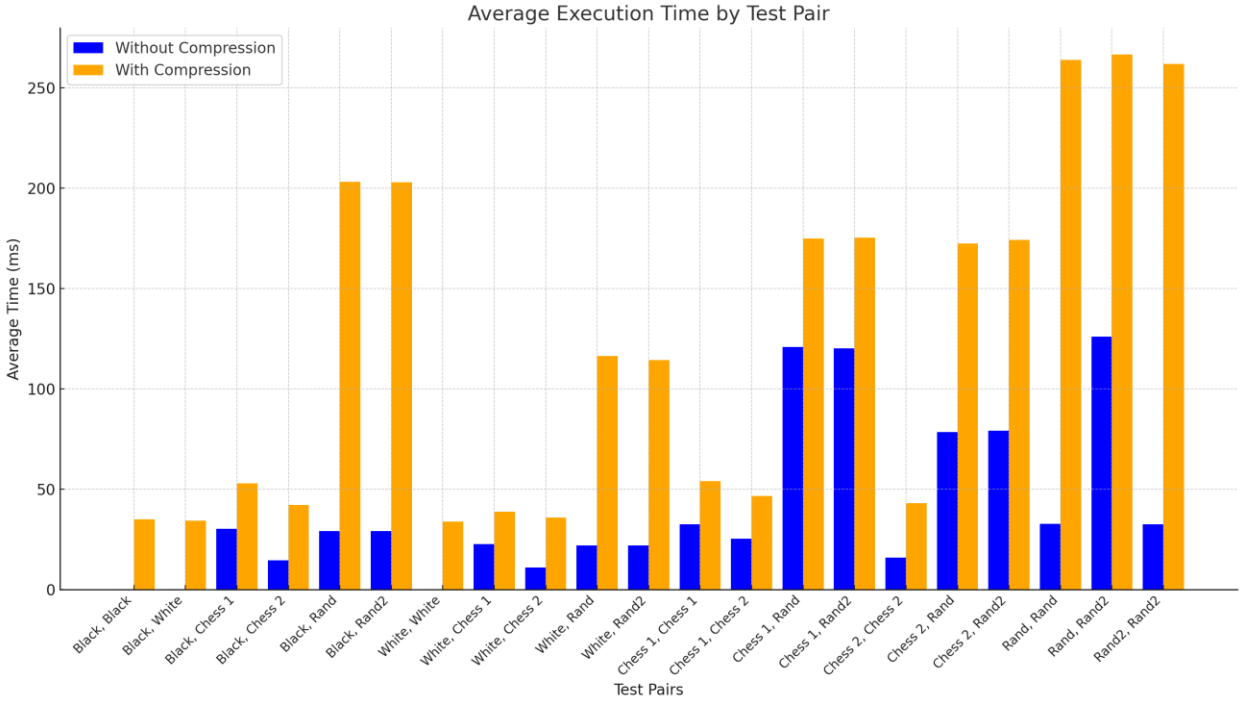
1. Alocação de memória para o cabeçalho da imagem de saída.
2. Processamento linha a linha das imagens de entrada.
3. Combinação de intensidades e comprimentos de execução (RLE).
4. Alocação e ajuste dinâmico do *array* resultante.

Dados experimentais

Tabela e Gráfico com todas as combinações possiveís entre 6 imagens 4096x4096, Imagem toda preta/branca, *chessboard* com lado 2 e 4, e 2 imagens geradas aleatoriamente com (rand() %2, pixel a). E os respetivos tempos execução da função AND

com e sem utilização (des)compressão (comp And e RLE AND, respetivamente), com cpu_time(). E respetivo gráfico.

Test Pair	RLE AND	Comp And	Test Pair	RLE And	Comp And
Black, Black	0.257	34.931	Chess1, Chess1	35.546	53.989
Black, White	0.234	34.205	Chess1, Chess2	25.305	46.598
Black, Chess1	30.364	52.880	Chess1, Rand1	120.919	174.811
Black, Chess2	14.641	42.147	Chess1, Rand2	120.182	175.357
Black, Rand1	29.156	203.150	Chess2, Chess2	15.919	42.987
Black, Rand2	29.164	202.994	Chess2, Rand1	78.408	172.431
White, White	0.209	33.943	Chess2, Rand2	79.235	174.240
White, Chess1	22.600	38.813	Rand1, Rand1	32.814	263.998
White, Chess2	10.870	35.805	Rand1, Rand2	125.959	266.588
White, Rand1	22.045	116.271	Rand2, Rand2	32.547	261.955
White, Rand2	21.951	114.281			



Através desta análise, permite concluir que a aplicação do AND diretamente do formato RLE, aumenta em muito a eficiência, especialmente com maiores segmentos RLE e mais regulares (ou seja, menor número de segmentos).

Análise Formal da Complexidade Computacional

Iteração sempre sobre a altura e realiza sempre um conjunto de operações e comparações a cada linha com tempo e complexidade computacional constante $O(1)$.

Altura: H ; Largura: W ; Número de Runs: k

Melhor Caso (B(n))

- Condição: As linhas são uniformes, ou seja, apenas possuem apenas uma cor ou um segmento RLE.
- Comportamento: O *loop* interno realiza apenas realiza uma iteração.
- $B(n) = H.$
- Complexidade: $T(n) = O(H),\ onde\ n = H \cdot W.$

Pior Caso (W(n))

- Condição: Alternância pixel a pixel da cor, a cada linha, (Linha RLE: $[X, 1,1, \dots, 1]$).
- Comportamento: O número de runs/elementos RLE é igual à largura das imagens, o *loop* interno percorre W elementos por linha
- $W(n) = H.W$
- Complexidade: $T(n) = O(H.W), onde\ n = H \times W.$

Caso Médio (A(n))

- Condição: Cada linha tem n segmentos RLE, e k é o maior número, entre as imagens recebidas nos argumentos, de segmentos por linha RLE.
- Comportamento: O *loop* interno percorre k segmentos por linha.
- Como há 2 possibilidades na cor do pixel, no caso médio, $k \approx \frac{W}{2},\ K < W$
- $A(n) = H.k$, logo $A(n) = \frac{H.W}{2}$
- Complexidade: $T(n) = O(H.W),\ onde\ n = H \times W.$

Espaço em memória

A cada iteração aloca um *array* com $W + 2$ elementos, e depois através de nova alocação fica a ocupar $k + 2$.

Cada linha da imagem resultante do algoritmo contem um *array* proporcional ao número de segmentos RLE. No pior caso são $W + 2$ elementos ($O(W)$).

Elementos são inteiros (de 4 bytes).

Análise Comparativa: Algoritmo Básico/Algoritmo Melhorado

Um algoritmo mais básico e fácil de implementar, é através do uso de descompressão, operação lógica pixel a pixel e nova compressão.

Estrutura e Complexidade (T(n)) do Algoritmo Básico

1. Iteração linha a linha - $O(H)$
2. Descompressão das duas imagens (dependente do número e tamanho de cada *run* RLE) - $O(W)$
3. Iteração pixel a pixel (operação de *AND*) - $O(W)$
4. Compressão (iteração a cada pixel da linha) - $O(W)$

Complexidade Temporal:

$$T(n) = H.(4.W)$$

$$T(n) = O(H.W)$$

Comparação

Ambos os algoritmos têm a mesma ordem de complexidade ($O(W.H)$). A sua complexidade temporal varia apenas na constante pelo qual são multiplicados.

$$A(n) = \frac{H.W}{2}$$

$$T(n) = H.(4.W)$$

Logo,

$$8 \cdot A(n) = T(n)$$

Do ponto de vista ocupação de espaço em memória são algoritmos idênticos, ocupando temporariamente um *array com* $W + 2$ elementos e guardando na nova imagem um *array* proporcional ao número de segmentos RLE.

Conclusão Final

Este trabalho permitiu o desenvolvimento de competências importantes na implementação, análise e aperfeiçoamento de algoritmos. Para tal foram feitos vários testes experimentais e análises de complexidade de modo a avaliar e aprimorar o desempenho das diferentes soluções algorítmicas. Este projeto demonstrou a importância de fazer escolhas fundamentadas no design de algoritmos e considerar diversos cenários de uso. A capacidade de comparar diferentes abordagens proporcionou o desenvolvimento de um pensamento crítico sobre as vantagens, desvantagens e limitações de cada solução, uma competência valiosa para futuros projetos desenvolvidos.