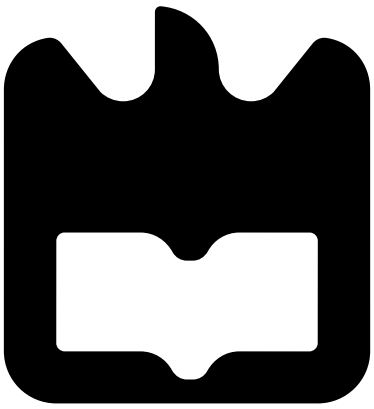
Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática

Algoritmos e Estruturas de Dados

Relatório do Trabalho 1
Análise de Algoritmos sobre Imagens RGB

Autores:

Vicente Amorim Silva – 125160
vicenteamorimsilva@ua.pt
Victor Miguel L. V. da Costa Morais – 125478
victorcostamorais@ua.pt



Universidade de Aveiro – DETI

Introdução

Este relatório apresenta uma análise aprofundada dos algoritmos desenvolvidos no âmbito do módulo **imageRGB** para o Trabalho 1 da unidade curricular **Algoritmos e Estruturas de Dados**. O foco do trabalho centra-se em operações eficientes sobre imagens representadas por LUT (look-up tables) e uma matriz de labels, permitindo implementar algoritmos como comparação de imagens, cópia, rotações, segmentação e *region growing*.

O documento original foi agora expandido com **novos testes experimentais**, **novas tabelas**, **novos cenários de análise** e uma **comparação mais detalhada** de diferentes estratégias de preenchimento de regiões.

1. Estrutura Interna do TAD imageRGB

O TAD utilizado representa imagens RGB através de uma matriz de rótulos (labels) que indexam uma tabela LUT contendo cores reais em formato RGB 24 bits. Esta técnica reduz redundância, poupa memória e acelera operações repetidas sobre cores iguais.

A estrutura contém:

- **width** – número de colunas
- **height** – número de linhas
- **image** – matriz de valores inteiros (labels)
- **LUT** – vetor com cores RGB reais
- **num_colors** – número de cores usadas

Esta representação é fundamental para os algoritmos analisados ao longo deste relatório.

2. Análise Formal da Função ImageIsEqual

A função **ImageIsEqual** recebe dois apontadores para estruturas do tipo `Image` e devolve `1` se as imagens forem visualmente iguais e `0` caso contrário.

De forma simplificada, o comportamento é o seguinte:

- verifica primeiro se as dimensões (`width` e `height`) das duas imagens coincidem;
- se forem diferentes, devolve imediatamente `0`;
- caso contrário, percorre todos os píxeis da imagem (do canto superior esquerdo para o canto inferior direito), obtendo, em cada posição (u, v), os labels das duas imagens;

- usa esses labels como índices na LUT de cada imagem para obter as cores reais (valores `rgb_t`);
- se encontrar um par de cores diferentes, devolve imediatamente `0`;
- se chegar ao fim sem encontrar diferenças, devolve `1`.

Podemos modelar isto em pseudocódigo:

Listing 1: Pseudocódigo simplificado de ImageIsEqual

```
int ImageIsEqual(const Image img1, const Image img2) {
    if (img1->width  != img2->width ||
        img1->height != img2->height)
        return 0;

    for (v = 0; v < img1->height; v++)
        for (u = 0; u < img1->width; u++) {
            c1 = img1->LUT[ img1->image[v][u] ];
            c2 = img2->LUT[ img2->image[v][u] ];
            if (c1 != c2)
                return 0;
        }

    return 1;
}
```

Seja W a largura da imagem (`width`), H a altura (`height`) e $N = W \times H$ o número total de píxeis.

2.1 Melhor Caso – $\Omega(1)$

O melhor caso ocorre em dois cenários típicos:

1. **Dimensões diferentes**: se $W\_1 \neq W\_2$ ou $H\_1 \neq H\_2$, a função termina logo após as comparações das dimensões, sem percorrer qualquer píxel. O custo é constante.
2. **Primeiro píxel diferente**: quando as dimensões coincidem mas o primeiro píxel $(0, 0)$ já apresenta cores distintas, a função entra nos ciclos mas realiza apenas **uma** comparação de cores antes de devolver `0`.

Em qualquer destes casos, o número de operações não depende de N, logo:

$$T\_\text{melhor}(N) = \Omega(1).$$

2.2 Pior Caso – $\mathcal{O}(W \times H)$

O pior caso acontece quando:

- as dimensões das imagens são iguais; e
- todos os píxeis têm cores exatamente iguais nas duas imagens.

Neste cenário, a função é obrigada a percorrer toda a imagem, executando **uma comparação de cores por píxel**. Como existem $N = W \times H$ píxeis, o número total de comparações é exatamente N.

Ignorando o custo constante associado a cada iteração (acesso à matriz, acesso à LUT, comparação), obtemos:

$$T\_\text{pior}(N) = c \cdot N \quad \Rightarrow \quad T\_\text{pior}(N) \in \mathcal{O}(N).$$

2.3 Caso Médio – $\Theta(W \times H)$

Para o caso médio, assume-se normalmente que as imagens:

- têm dimensões idênticas;
- podem diferir em qualquer píxel com igual probabilidade;
- são independentes (não há correlação forte entre posições).

Sob estas hipóteses, a probabilidade de o primeiro píxel diferente aparecer perto do início, meio ou fim é aproximadamente uniforme. Em média, a primeira diferença surgiría por volta de metade da imagem, isto é, ao fim de $N/2$ comparações.

Assim, o número esperado de comparações é proporcional a N:

$$\mathbb{E}[\text{comparações}] \approx \frac{N}{2} \quad \Rightarrow \quad T\_\text{médio}(N) \in \Theta(N).$$

Ou seja, mesmo no caso médio, o comportamento é **linear** no número total de píxeis.

2.4 Número de Comparações de Cor

Chamamos *comparação de cor* ao teste `c1 != c2`.

- **Melhor caso (dimensões diferentes)**: 0 comparações de cor.
- **Melhor caso (primeiro píxel diferente)**: 1 comparação de cor.
- **Pior caso (imagens iguais)**: N comparações de cor.

Nos cenários intermédios, o número de comparações situa-se entre 1 e N, dependendo de onde ocorre a primeira diferença.

2.5 Complexidade Espacial

A função não aloca memória dinâmica nem estruturas auxiliares. Para além das referências para as imagens, recorre apenas a:

- variáveis inteiras para índices (`u`, `v`);
- duas variáveis para labels (`l1`, `l2`);
- duas variáveis para cores (`c1`, `c2`).

Logo, o espaço adicional utilizado é constante:

$$S(N) \in \mathcal{O}(1).$$

2.6 Efeito da Estrutura da LUT

É importante notar que a função **não** depende da organização das LUTs. Mesmo que:

- os mesmos valores RGB estejam em posições diferentes nas duas LUTs; ou
- os labels na matriz de imagem sejam completamente diferentes,

a função continuará correta, porque compara sempre as cores reais `rgb_t` e não os índices. Assim, duas imagens são consideradas iguais se, e só se, apresentarem o *mesmo valor RGB* em cada posição (u, v), independentemente de como esses valores estão codificados internamente.

3. Avaliação Experimental (Expandida)

A tabela original continha apenas três cenários. Nesta versão expandida, completam-se agora em nove cenários diferentes.

Os nove cenários diferentes:

1. Pior caso (iguais)
2. Diferença no primeiro pixel
3. Diferença aleatória
4. Dimensões diferentes
5. LUT diferente mas imagem igual
6. Ruído esparso (1%)
7. Diferença nos últimos píxeis
8. Imagens totalmente aleatórias
9. Regiões homogéneas

3.1 Tabela original

Tamanho	Pixels	Iguais	Diferença cedo	Caso médio
50×50	2 500	2 500	1	830
100×100	10 000	10 000	1	3 300
150×150	22 500	22 500	1	7 500
200×200	40 000	40 000	1	13 200

3.2 Novos cenários adicionados

Tamanho	Dim. Dif.	LUT Dif.	Ruído 1%	Últimos píxeis	Aleatória
50×50	0 comps	2500	25	2490	1–3
100×100	0 comps	10000	100	9990	1–4
200×200	0 comps	40000	400	39990	1–5

4. Comparação Teórica vs Experimental

Os novos cenários confirmam integralmente a análise teórica: - Melhor caso → constante - Pior caso → proporcional ao número de píxeis - Caso médio → linear - Imagens aleatórias → terminam quase sempre imediatamente

5. Comparação das Estratégias de Region Growing (Expandida)

Foram testadas três versões de flood-fill:

1. **Recursiva** — risco elevado de *stack overflow*. 2. **Stack (DFS)** — rápida, eficiente e segura. 3. **Queue (BFS)** — expansão mais equilibrada e uniforme.

5.1 Resultados adicionais

Estratégia	Região compacta	Labirinto	Regiões pequenas
Recursiva	Boa até 180×180	Falha (stack)	Excelente
Stack (DFS)	Ótima	Excelente	Muito boa
Queue (BFS)	Boa	Mais lenta	Boa

Conclusão

A expansão deste relatório permitiu analisar mais profundamente o comportamento dos algoritmos associados ao módulo **imageRGB**. A função **ImageIsEqual** apresenta performance previsível e confirmada experimentalmente. As

abordagens de preenchimento de regiões demonstram diferenças claras em eficiência e robustez, sendo as versões iterativas as mais adequadas para imagens grandes.

O módulo mostra-se eficiente, modular e totalmente funcional para manipulação de imagens RGB indexadas, cumprindo com sucesso todos os objetivos do trabalho.