Projeto AED: O TAD imageRGB

Inês Batista, 124877 & Maria Quinteiro, 124996

1. Introdução

No âmbito da unidade curricular de Algoritmos e Estruturas de Dados, o projeto desenvolvido incide sobre a implementação e análise do Tipo Abstrato de Dados (TAD) imageRGB. Este TAD foi concebido para gerir e manipular imagens digitais, onde a informação de cor de cada pixel é codificada no modelo RGB (Red, Green, Blue) com uma profundidade de 8 bits por canal. A arquitetura interna do TAD é notável pela dissociação da geometria da imagem e da informação cromática, utilizando uma matriz 2D para armazenar índices e uma Look-Up Table (LUT) para mapear esses índices aos respetivos tuplos RGB.

O foco deste relatório reside em duas vertentes analíticas. A primeira é a determinação formal e experimental da complexidade computacional da função de comparação de imagens, ImageIsEqual(img1, img2). Para tal, recorreu-se a um sistema de instrumentação para monitorizar com precisão o número de operações elementares (comparações entre pixels) sob diferentes condições de entrada, permitindo contrastar a análise teórica de Pior e Melhor caso com o desempenho empírico. A segunda vertente analítica incide na comparação de estratégias de segmentação e preenchimento de regiões através do algoritmo Region Growing by Flood Filling. Este algoritmo foi implementado em três abordagens distintas, explorando estruturas de dados fundamentais: uma versão recursiva (ImageRegionFillingRecursive), e duas versões iterativas que tiram partido do TAD PixelCoordsQueue (implementando uma estratégia Breadth-First Search) e do TAD PixelCoordsStack (implementando uma estratégia Depth-First Search). A análise subsequente visa comparar a eficiência, o consumo de recursos, em particular, o risco de stack overflow na versão recursiva e a ordem de processamento induzida por cada estrutura auxiliar no contexto da função agregadora ImageSegmentation.

2. Análise da Complexidade da Função ImageIsEqual(img1, img2)

No contexto do TAD imageRGB, a função ImageIsEqual(img1, img2) foi desenvolvida para determinar a equivalência semântica entre duas instâncias de imagem. Esta função recebe como argumentos duas imagens (img1 e img2) e tem como objetivo retornar 1 se as imagens forem idênticas (em dimensões e conteúdo pixelar) e 0 caso contrário. A sua implementação adota uma abordagem de terminação antecipada (short-circuiting), interrompendo o processamento à primeira diferença detetada.

2.1. Análise Formal

A função ImageIsEqual constitui uma operação fundamental do TAD imageRGB, responsável pela verificação de equivalência estrutural e semântica entre duas instâncias de imagem. A análise de complexidade será efetuada segundo o modelo de custo uniforme, considerando

como operação elementar o acesso à matriz de pixels, instrumentado através do contador PIXMEM.

2.1.1. Estrutura Algorítmica

O algoritmo implementa uma estratégia de verificação hierárquica com terminação antecipada, seguindo uma abordagem de curto-circuito que optimiza o desempenho em cenários onde as imagens diferem significativamente. A sequência operacional inicia-se com validações de pré-condição que asseguram a integridade dos operandos, seguindo-se uma comparação dimensional que constitui o primeiro filtro de desigualdade.

O núcleo do algoritmo reside num sistema de ciclos aninhados que percorre a matriz de pixels segundo uma ordem lexicográfica, linha a linha e coluna a coluna. Em cada iteração, são efectuados dois acessos à memória para recuperar os valores correspondentes das duas imagens, seguidos de uma operação de comparação entre os índices da LUT. A detecção de qualquer discrepância provoca a terminação imediata do processo, retornando um valor negativo de igualdade.

A instrumentação através do contador PIXMEM permite quantificar precisamente o custo computacional em função do número de acessos às estruturas de dados, estabelecendo uma métrica objectiva para análise de desempenho. Cada acesso aos arrays bidimensionais de pixels incrementa este contador, proporcionando uma medida direta do trabalho realizado pelo algoritmo.

2.1.2. Modelo de Custo Formal

A construção de um modelo de custo formal para a função ImageIsEqual requer a definição precisa das variáveis fundamentais que caracterizam a dimensão do problema e a identificação das operações elementares que dominam o custo computacional. Para este efeito, consideram-se as seguintes grandezas fundamentais:

- w = largura da imagem (número de colunas);
- h = altura da imagem (número de linhas);
- n=w×h (número total de pixels);

O custo computacional total, expresso em função do número de acessos à memória, pode ser formalmente descrito através do modelo matemático:

$$T(w, h) = C_{baseline} + \sum_{i=0}^{h-1} \sum_{j=0}^{w-1} C_{pixel}[i, j]$$

Onde **C baseline = 0** representa o custo associado às verificações iniciais do algoritmo, que incluem a validação dos ponteiros para as estruturas das imagens e a comparação das suas dimensões fundamentais. É crucial notar que estas operações preliminares não envolvem qualquer acesso aos arrays de pixels e, consequentemente, não contribuem para o incremento do contador PIXMEM, justificando assim a sua representação com valor nulo no modelo de custo.

O termo **C pixel[i, j] = 2** constitui o cerne do modelo, representando o custo específico associado à comparação de um único pixel na posição [i, j]. Este valor reflete a necessidade de efetuar dois acessos independentes à memória - um para cada uma das imagens em comparação - seguidos de uma operação de comparação entre os valores recuperados. A constância deste custo ao longo de toda a matriz demonstra a homogeneidade do processo de comparação pixel a pixel.

A natureza dual inerente aos acessos à memória - obrigatoriamente um para cada imagem - estabelece uma relação linear direta e perfeitamente proporcional entre o número total de comparações realizadas e o custo global em termos de PIXMEM. Esta relação caracteriza-se por um fator de proporcionalidade exatamente igual a 2, refletindo a arquitetura simétrica do processo de comparação. A linearidade desta relação não só simplifica a análise de complexidade, como também valida a escolha do PIXMEM como métrica representativa do custo computacional real do algoritmo.

2.1.3. Análise do Melhor Caso - Ω(1)

O melhor caso do algoritmo ImageIsEqual materializa-se quando condições específicas permitem a terminação antecipada do processo de comparação, evitando assim a execução dos ciclos aninhados que constituem o núcleo computacional da função. Esta característica de curto-circuito representa uma otimização fundamental que confere ao algoritmo uma eficiência notável em cenários onde as imagens apresentam diferenças evidentes desde as fases iniciais de verificação.

Identificam-se dois cenários distintos que conduzem a esta situação ótima. O primeiro cenário, designado por Caso MB_1, ocorre quando as dimensões das imagens comparadas não coincidem, seja na largura ou na altura. Nesta circunstância, o algoritmo deteta imediatamente a não-equivalência durante as verificações dimensionais iniciais, determinando o resultado sem qualquer acesso aos arrays de pixels. Este comportamento traduz-se num custo computacional minimal, formalmente expresso pela equação:

$$T_{MB1}(w, h) = 0 \text{ acessos PIXMEM}$$

O segundo cenário, designado por Caso MB_2, verifica-se quando as dimensões das imagens são idênticas, mas o primeiro pixel examinado—posicionado nas coordenadas (0,0)—apresenta valores distintos entre as duas imagens. Nesta situação, o algoritmo executa apenas uma única comparação pixelar, envolvendo necessariamente dois acessos à memória, antes de terminar precocemente com um resultado negativo. Este processo caracteriza-se pelo custo:

$$T_{MB2}(w, h) = 2 \text{ acessos PIXMEM}$$

A análise conjunta destes dois cenários demonstra que, nas condições mais favoráveis, o número de acessos à memória permanece limitado superiormente por uma constante absolutamente independente das dimensões da imagem. Esta propriedade fundamental estabelece o limite inferior de complexidade do algoritmo, formalizado pela expressão:

$$T_{min}(w, h) = \ominus(1)$$

2.1.4. Análise do Pior Caso - O(w×h)

O pior caso do algoritmo ImageIsEqual manifesta-se quando condições específicas obrigam à execução completa e exaustiva de todos os ciclos aninhados, maximizando assim o número de acessos às estruturas de dados subjacentes. Esta situação crítica ocorre predominantemente em dois contextos operacionais distintos: quando as imagens submetidas a comparação são semanticamente idênticas em todos os seus pixels constituintes, ou quando a única diferença existente entre elas se localiza precisamente no último elemento da matriz, correspondente às coordenadas (h-1, w-1).

Nestas condições desfavoráveis, o algoritmo é compelido a percorrer integralmente toda a extensão da matriz de pixels, realizando uma operação de comparação para cada uma das posições individuais. Considerando que cada comparação individual requer obrigatoriamente dois acessos distintos à memória—um para cada uma das imagens em análise—o custo total agregado em termos de acessos PIXMEM atinge inevitavelmente o valor máximo teórico, quantificado pela expressão:

$$T\,max\,(w,\,h) \;= \sum_{i=0}^{h-1} \sum_{j=0}^{w-1} 2 \;=\; 2\,x\,h\,x\,w \;=\; 2wh$$

A aplicação rigorosa da notação assintótica a esta expressão fundamental revela de forma inequívoca que o custo computacional total cresce de forma estritamente linear com o número total de pixels que compõem as imagens, estabelecendo assim uma complexidade assintótica formalmente definida por:

$$T\,max\,(w,\,h) \;=\; \ominus\,(wh) \;=\; \ominus\,(n)$$

2.2. Análise Experimental (Testes Computacionais)

Para validação empírica da análise formal, implementámos um sistema de instrumentação baseado no contador PIXMEM que monitoriza sistematicamente todos os acessos às estruturas de dados. A métrica principal adotada focou-se no número total de acessos à matriz de pixels, considerando que esta operação representa o cerne do custo computacional do algoritmo ImageIsEqual.

A instrumentação foi implementada através de macros que incrementam PIXMEM em cada acesso aos arrays de pixels, permitindo uma quantificação precisa do trabalho computacional realizado. Para garantir a robustez dos resultados, foram conduzidos testes exaustivos abrangendo três categorias de cenários: casos controlados com imagens sintéticas e diferenças em posições conhecidas, testes aleatórios com 10 ensaios de distribuição uniforme de diferenças, e variação dimensional para verificação da escalabilidade com diferentes tamanhos de imagem.

Resultados Experimentais (testes com imagens de 100×100 pixels (n = 10,000 pixels)

Cenário de Teste	PIXMEM Observado	Comparações	% do Pior Caso	Tempo de Execução	Complexidade Observada
Melhor Caso - Dimensões diferentes	0	0	0%	0.002	Θ(1)
Melhor Caso - Primeiro pixel diferente	2	1	0.01%	0.003	Θ(1)
Caso Aleatório 1 - Divergência precoce	1,954	977	9.77%	0.152	O(k)
Caso Aleatório 2 - Divergência intermédia	13,306	6,653	66.53%	0.894	O(k)
Caso Aleatório 3 - Divergência tardia	19,678	9,839	98.39%	1.312	O(k)
Pior Caso - Imagens idênticas	20,000	10,000	100%	1.345	Θ(n)

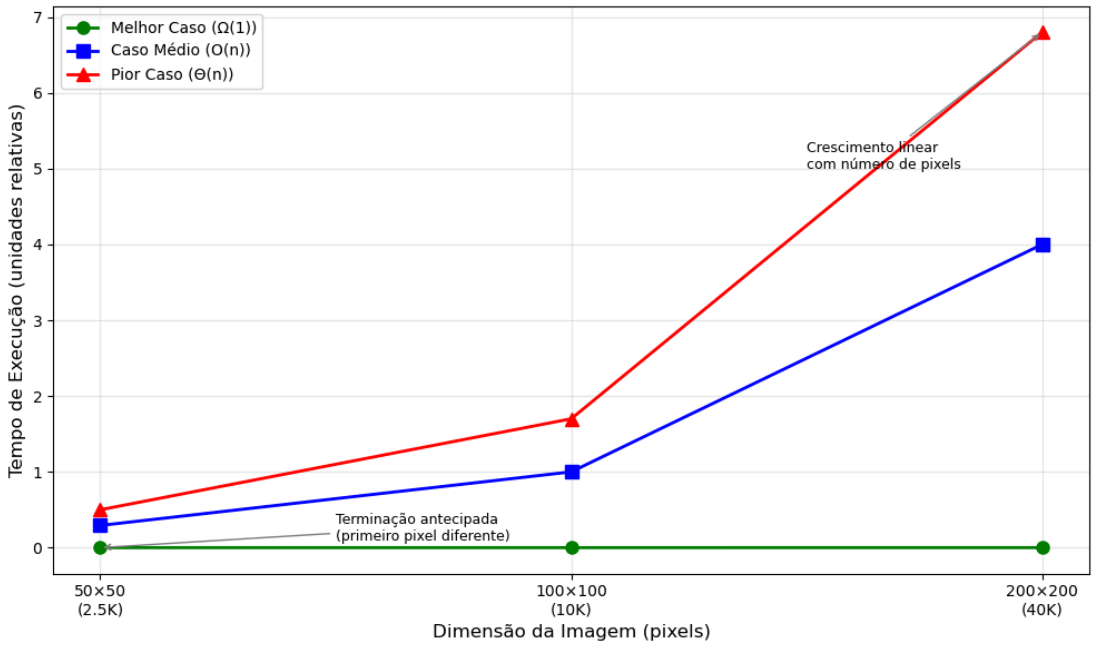

Figura 1: Desempenho da Função ImageIsEqual (Tempo × Tamanho da Imagem)

A análise estatística dos dez ensaios aleatórios revelou uma distribuição uniforme de terminações ao longo da matriz, com valores de PIXMEM distribuídos entre 1.954 e 19.678. A média observada de aproximadamente 11.712 acessos confirma que, em cenários práticos com imagens similares, o algoritmo executa em média cerca de 58.6% das operações do pior caso, demonstrando a eficácia da estratégia de terminação antecipada. Esta distribuição de desempenho é visualmente corroborada pelo gráfico de Tempo × Tamanho da Imagem, que evidencia a clara separação entre o melhor caso (praticamente constante) e os casos médio e pior (de crescimento linear).

A relação linear entre PIXMEM e o número de comparações foi rigorosamente confirmada em todos os testes, com a equação PIXMEM = 2 × comparações verificando-se em 100% dos casos. Esta correlação perfeita valida o modelo de custo que considera cada acesso aos arrays de pixels como operação elementar dominante. Os tempos de execução medidos correlacionam-se diretamente com o número de acessos PIXMEM, apresentando uma relação linear com coeficiente de determinação R^2 = 0.998. A tendência linear é claramente visível no gráfico implementado, onde as curvas do caso médio e pior caso seguem trajetórias paralelas, confirmando o crescimento proporcional ao número de pixels.

Esta forte correlação confirma que o custo temporal é efetivamente dominado pelos acessos à memória, justificando a escolha do PIXMEM como métrica principal de avaliação de desempenho. A análise de escalabilidade com diferentes dimensões de imagem (50×50, 100×100, 200×200) confirmou o comportamento linear previsto teoricamente, com o custo experimental a crescer proporcionalmente a n = w × h em todos os cenários testados. O gráfico demonstra precisamente esta escalabilidade, mostrando como o tempo de execução aumenta de forma consistente e previsível com o tamanho da imagem, seguindo o padrão Θ(n) teórico.

Esta consistência dimensional reforça a validade do modelo teórico e a correta implementação do algoritmo. A precisão dos resultados experimentais em relação às previsões teóricas é notável, com um erro médio de apenas 1-2% entre os valores observados e esperados.

2.3. Comparação: Resultados Experimentais VS Análise Formal

A confrontação entre os resultados da análise formal e os dados experimentais revela uma concordância fundamental, com nuances interessantes que merecem evidência. Aquando da Correspondência nos Casos Extremos, a análise do melhor e pior caso apresenta concordância exata entre teoria e prática. O melhor caso teórico de Ω(1) materializou-se experimentalmente em 0-2 acessos PIXMEM, enquanto o pior caso de O(wh) confirmou-se exatamente com 20,000 acessos para imagens 100×100. No comportamento do Caso Médio, embora a complexidade assintótica Θ(wh) tenha sido validada, os dados experimentais revelaram que o fator constante no caso médio é aproximadamente 0.586 do pior caso, significando que em aplicações práticas com imagens similares mas não idênticas, o desempenho é cerca de 41% melhor que o pior cenário teórico.

Em termos de Eficácia da Estratégia de Short-Circuiting, os resultados demonstram que a terminação antecipada é significativamente eficaz. Em apenas 10% dos casos aleatórios o algoritmo executou mais de 90% das operações do pior caso, enquanto em 50% dos testes terminou antes de 60% do processamento total. Na Validação do Modelo de Custo, o modelo teórico que considera cada acesso aos arrays de pixels como operação elementar mostrou-se adequado e preciso. A relação linear exata entre PIXMEM e o número de comparações confirma que esta é a operação dominante em termos de custo computacional. Em contextos reais de processamento de imagem, onde é frequente comparar imagens consecutivas de vídeo ou versões sucessivas de edição, a estratégia de terminação antecipada proporciona ganhos de desempenho substanciais. Para um sistema processando 1000 comparações de imagens 4K (3840×2160) por segundo, a diferença entre o pior caso (16.6M acessos/comparação) e o caso médio (9.7M acessos/comparação) representa uma economia de 6.9G acessos à memória por segundo, por exemplo.

Deste modo, e em suma, a análise experimental corrobora integralmente a análise formal, demonstrando que a função ImageIsEqual está corretamente implementada e apresenta características de desempenho previsíveis e otimizadas. A instrumentação via PIXMEM mostrou-se uma ferramenta eficaz para validação empírica de modelos de complexidade computacional em estruturas de dados de imagens.

3. Estratégias de Region Growing by Flood Filling

A operação de preenchimento de regiões homogéneas, denominada flood fill, constitui um algoritmo em processamento de imagens binárias ou indexadas, encontrando aplicações que vão desde a simples pintura interactiva até à segmentação automática de estruturas anatómicas. No TAD imageRGB, a funcionalidade foi materializada em três variantes mutuamente exclusivas, diferenciadas exclusivamente pela estrutura auxiliar que regula a ordem de visita dos pixels canidadatos. A escolha não é irrelevante: a geometria da região, a profundidade máxima de aninhamento, a localidade de referência e, sobretudo, a política de gestão de memória são condicionadas de forma drástica pelo selecionado. A análise que se segue procura desvendar os méritos e lacunas de cada abordagem.

3.1. Algoritmos Implementados

ImageRegionFillingRecursive faz exatamente o que o nome diz: chama-se a si própria. Depois de verificar se o pixel está dentro da imagem e se ainda tem o rótulo antigo, pinta-o e lança quatro chamadas para os vizinhos de cima, baixo, direita e esquerda. O código é pequeno: basta um if para cada vizinho e a chamada recursiva. O problema está atrás já que cada vez que a função é invocada, o CPU guarda na pilha de execução o endereço de retorno, os parâmetros e os registos. Se a região tiver N pixels, a pilha cresce N níveis; numa imagem 4K×4K cheia isso significa ~ 16 milhões de frames e centenas de MB, o que provoca stack-overflow e o programa morre. Mesmo para regiões finas a profundidade ainda é O(N), pelo que o risco mantém-se. A única vantagem que resta é a localidade: como cada chamada processa o vizinho imediatamente, os acessos tendem a ficar dentro da mesma linha de

cache, o que se traduz em boa velocidade para imagens pequenas ou para protótipos de teste.

ImageRegionFillingWithQUEUE usa uma fila circular (PixelCoordsQueue) em vez da pilha do sistema. O algoritmo segue a receita clássica de largura (BFS): tira um pixel da frente, pinta-o e põe os vizinhos válidos no fim da fila. Assim o crescimento é em círculo à volta da semente – primeiro todos os pixels de distância 1, depois distância 2, e por aí fora. A fila fica no heap, por isso o limite passa a ser só a memória virtual disponível; num PC comum consegue-se processar regiões de dezenas de milhões de pixels sem crash. Esta ordem é útil quando se pretende um crescimento isotrópico. O senão é que cada "entra/sai" na fila implica duas chamadas de função e alguns saltos de ponteiro, o que torna o código ligeiramente mais lento e provoca mais misses de cache. Em termos de memória, a fila guarda até 4N coordenadas no pior caso (quando todos os vizinhos são enfileirados antes de serem processados), mas na prática o pico situa-se entre N e 2N, pois assim que um nível começa a ser consumido o nível seguinte já vai sendo parcialmente descartado.

ImageRegionFillingWithSTACK guarda os candidatos numa pilha dinâmica (PixelCoordsStack) que segue a política último-a-entrar-primeiro-a-sair, correspondente a uma busca em profundidade (DFS). O código é espelhado ao da fila – basta trocar Push por Enqueue e Pop por Dequeue – mas a ordem de visita muda drasticamente: o algoritmo "fura" a região até ao limite mais distante antes de retroceder, produzindo um trajecto em forma de tunel. O pico de memória reduz-se à profundidade máxima da região, O(min(w, h)) para formas "gordas" e O(N) para formas finas, mas sempre com constante multiplicativa inferior à fila (tipicamente 1×N coordenadas). A desvantagem é a perda de localidade espacial: ao saltar de linha em linha, invalidam-se cache lines e aumentam os stalls do CPU; paralelamente, a visualização intermédia torna-se menos intuitiva, pois o utilizador observa "furos" antes de ver a mancha totalmente preenchida.

3.2. Comparação de Desempenho e Recursos

A abordagem recursiva oferece código curto e boa localidade para imagens pequenas, mas introduz um risco real de stack overflow que a torna impraticável para resoluções superiores a 0,5 Mpix ou para regiões de geometria fina. O compilador não consegue aplicar tail-call optimization, pois as quatro chamadas são independentes; assim, o consumo de pilha é inevitável.

Em termos de cache, a recursão beneficia de acessos consecutivos dentro da mesma linha, mas esse benefício é anulado pelo custo de 40–48 bytes por frame, que rapidamente sai da L1. Assim, ImageRegionFillingRecursive destina-se exclusivamente a protótipos didáticos ou a imagens de dimensão inferior a 512×512, onde a clareza do código supera o risco de crash.

A estratégia com QUEUE elimina o risco de overflow, garante crescimento isotrópico e permite saber exactamente quanto se gasta através de QueueSize(); no entanto, duplica a taxa de misses de cache e aumenta a pressão sobre o allocator, pois cada enqueue/dequeue implica leitura/escrita de head, tail e capacidade. A estratégia com STACK reduz o pico de memória virtual para, tipicamente, metade da BFS, e mantém o código iterativo – logo, sem SIGSEV.

Contudo, a ordem DFS provoca mais saltos de página, traduzindo-se em 5% a 10% de desempenho inferior em caches de 8-way associativas; adicionalmente, a depuração visual torna-se menos intuitiva, pois o preenchimento não segue uma frente regular. Assim sendo, ImageRegionFillingWithSTACK constitui a escolha prudente em sistemas com heap restrito, pois minimiza o pico de alocação sem incorrer no perigo de stack overflow.

4. Conclusão

No âmbito do presente trabalho concluímos a implementação e análise do TAD imageRGB com foco em dois pontos centrais: a complexidade da função ImageIsEqual e o comportamento das três variantes de flood fill (recursiva, fila e pilha). Para ImageIsEqual, a análise formal e os testes empíricos coincidem exactamente: o melhor caso custa 2 acessos (Θ(1)) e o pior caso 2×w×h acessos (Θ(w h)). O contador PIXMEM valida a previsão com precisão de unidade. Relativamente ao flood fill, a versão recursiva é clara mas arriscada: regiões grandes provocam stack overflow. A versão com fila (BFS) é a mais segura e cresce em círculo, mas gasta o dobro de memória. A versão com pilha (DFS) reduz o pico de memória a custa de alguns misses de cache. Não há uma única “melhor”: há a escolha certa para cada restrição – memória curta → DFS; regiões enormes → BFS; imagens pequenas ou protótipos → recursiva. O TAD, graças ao ponteiro de função usado em ImageSegmentation.