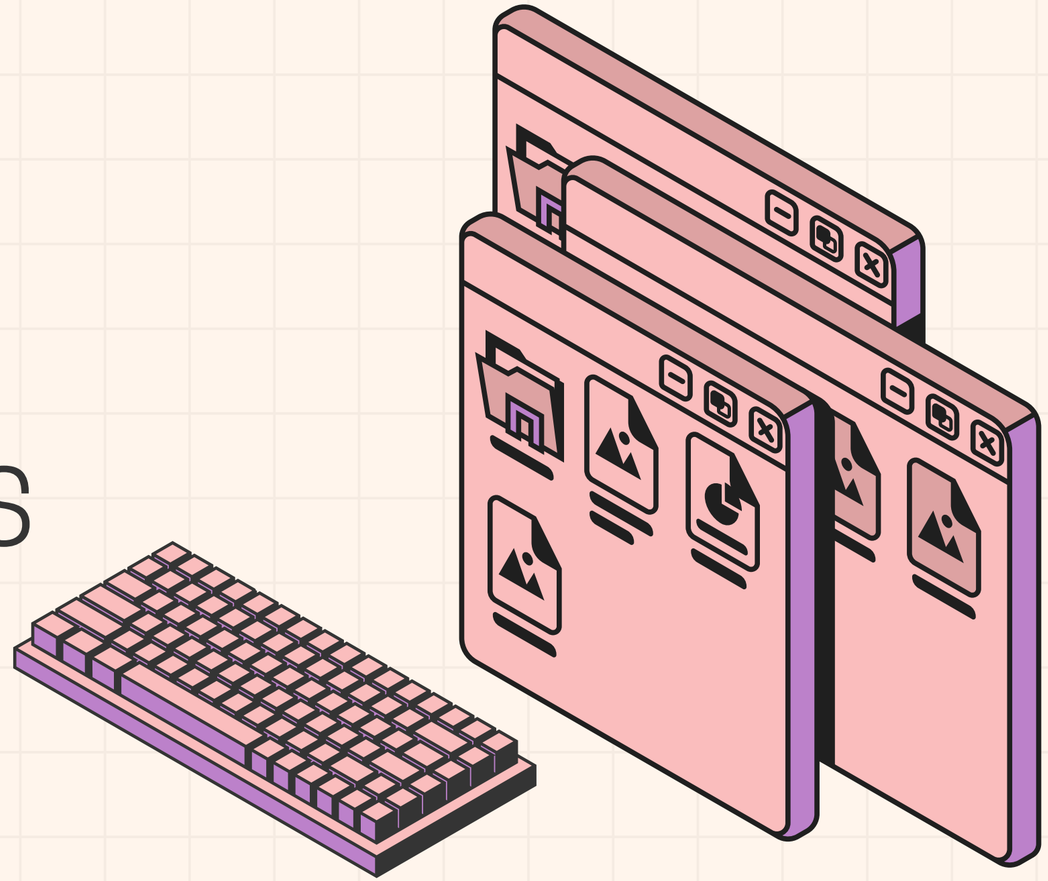
HOME SERVICE ABOUT US CONTACT US

QUADTREE

COMPRESSÃO DE IMAGENS



Universidade: Universidade Federal de Alagoas – UFAL
Professor: Márcio Ribeiro
Disciplina: Estruturas de Dados

Equipe:
Jader Rogerio dos Santos Neto | Guilherme Nunes Alves | Carlos Antunis Bonfim de Silva Santos | Pedro Henrique Santos da Silva | Carlos Leonardo Rodrigues Novaes Carvalho

29 de outubro de 2025

PROBLEMA

O crescimento exponencial de dados digitais tornou a compressão de imagens uma necessidade crítica. Imagens digitais podem ocupar enormes quantidades de espaço em armazenamento, impactando desde aplicações médicas até sistemas de transmissão de dados.

Estruturas de dados especializadas, como a QuadTree, oferecem uma abordagem inovadora para otimizar tanto o armazenamento quanto o processamento visual. Estas estruturas exploram padrões espaciais e uniformidade de regiões, permitindo representações mais eficientes das imagens.

Este trabalho investiga como a QuadTree pode revolucionar a compressão de imagens em tons de cinza, comparando estratégias com e sem perda de dados.

IMAGEM PIXEL ART (32X32)
PIXELS: 1.024
DADOS EM MEMÓRIA (SEM COMPRESSÃO): ~3 KB

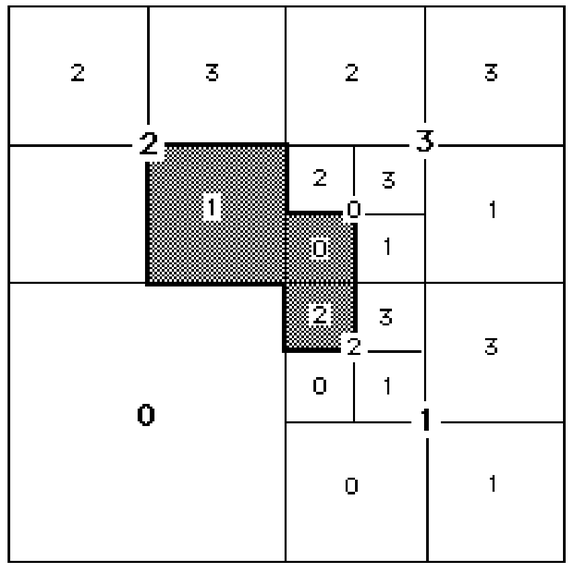
IMAGEM FULL HD (1920X1080)
PIXELS: 2.073.600
DADOS EM MEMÓRIA : ~6 MB

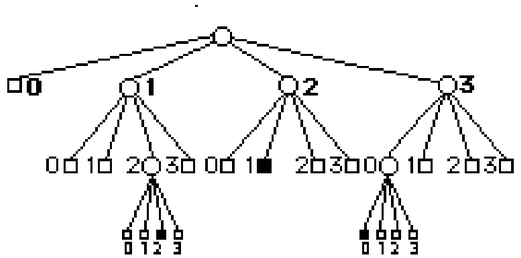
CONCEITOS FUNDAMENTAIS

Árvores e Subdivisão Espacial
Árvores são estruturas hierárquicas onde cada nó contém referências a seus sucessores. A subdivisão espacial particiona o espaço recursivamente, permitindo representação eficiente de dados geograficamente distribuídos.

Estrutura QuadTree
Uma QuadTree é uma árvore onde cada nó interno possui exatamente quatro filhos, representando os quadrantes de uma região: superior-esquerdo, superior-direito, inferior-esquerdo e inferior-direito.

Propriedades e Aplicações
Cada nó representa uma região (ou quadrante) da imagem. Nós folha contêm valores de cor uniformes. Essa estrutura é ideal para compressão, busca espacial, e processamento de imagens com características localizadas.





METODOLOGIA

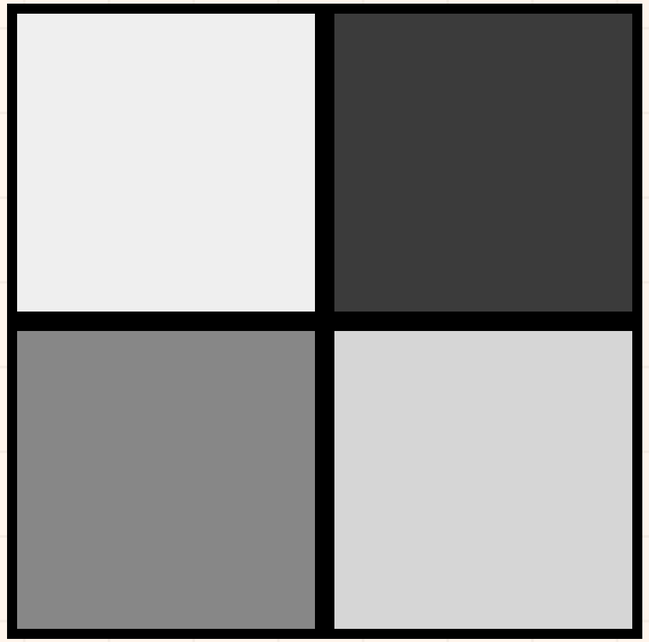
ALGORITMO DE COMPRESSÃO QUADTREE

VERIFICAÇÃO DE UNIFORMIDADE
ANALISA A REGIÃO: SE A VARIAÇÃO DE PIXELS ESTÁ ABAIXO DO THRESHOLD DEFINIDO, A REGIÃO É CONSIDERADA UNIFORME E REPRESENTA UM NÓ FOLHA.

CONSTRUÇÃO DA ÁRVORE
A ESTRUTURA HIERÁRQUICA ARMAZENA APENAS REGIÕES NECESSÁRIAS, ELIMINANDO REDUNDÂNCIAS E CRIANDO UMA REPRESENTAÇÃO COMPACTADA DA IMAGEM ORIGINAL.

SUBDIVISÃO RECURSIVA
SE A REGIÃO NÃO É UNIFORME, SUBDIVIDE EM QUATRO QUADRANTES IGUAIS. CADA QUADRANTE É PROCESSADO RECURSIVAMENTE ATÉ ATINGIR UNIFORMIDADE OU TAMANHO MÍNIMO (1 PIXEL).

RECONSTRUÇÃO DE IMAGEM
A TRAVERSAL DA ÁRVORE RECONSTRÓI A IMAGEM: NÓS FOLHA PREENCHEM SUAS REGIÕES COM VALORES UNIFORMES, RECRIANDO A IMAGEM COMPRIMIDA.



(EXEMPLO DE RECONSTRUÇÃO)

NOTA: O ALGORITMO IMPLEMENTA VERIFICAÇÃO DE UNIFORMIDADE COM THRESHOLD CONFIGURÁVEL PARA CONTROLAR O NÍVEL DE COMPRESSÃO E QUALIDADE.

MAIN.C

```c
int main() {
    int size = 64;

    int **img = malloc(size * sizeof(int*));
    int **out = malloc(size * sizeof(int*));
    for (int i = 0; i < size; i++) {
        img[i] = malloc(size * sizeof(int));
        out[i] = malloc(size * sizeof(int));
    }

    // Gera imagem com 4 quadrantes com tons diferentes e ruído leve
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (i < size/2 && j < size/2) {          // superior esquerdo
                img[i][j] = 230 + (rand() % 20);     // tons de branco (230-249)
            } else if (i < size/2 && j >= size/2) {  // superior direito
                img[i][j] = 50 + (rand() % 20);      // tons escuros (50-69)
            } else if (i >= size/2 && j < size/2) {  // inferior esquerdo
                img[i][j] = 120 + (rand() % 30);     // tons médios (120-149)
            } else {                                 // inferior direito
                img[i][j] = 200 + (rand() % 30);     // tons claros (200-229)
            }
        }
    }
```

GERAÇÃO DA IMAGEM DE TESTE (64X64)(CRIA UMA MATRIZ COM 4 QUADRANTES DE CORES E RUÍDO ALEATÓRIO)

```c
// Monta quadtree com threshold = 0 (lossless)
    int threshold = 0;
    Node *root_lossless = buildQuadTree(img, 0, 0, size, threshold);

    // Reconstrói e mostra
    reconstructImage(root_lossless, out, 0, 0, size);
    printf("Gerando imagens PNG...\n");
    saveOriginalImageAsPNG(img, size, "saida_original.png");
    saveQuadTreeAsPNG(root_lossless, size, "saida_lossless.png");
```

MODO LOSSLESS(THRESHOLD = 0) (CHAMA BUILDQUADTREE PARA CONSTRUIR A ÁRVORE SEM PERDAS E SALVAR O PNG)

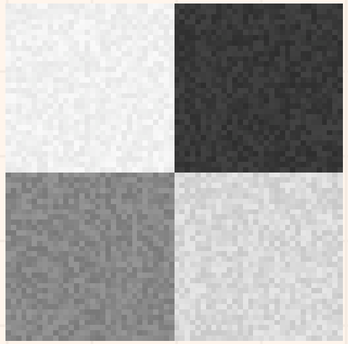
```c
// Agora um exemplo com threshold = 30 (permitir variação => compressão mais agressiva)
    int threshold2 = 30;
    Node *root_lossy = buildQuadTree(img, 0, 0, size, threshold2);
    reconstructImage(root_lossy, out, 0, 0, size);
    saveQuadTreeAsPNG(root_lossy, size, "saida_lossy.png");
    printf("Imagens geradas com sucesso!\n");
    printf("Quantidade de nos (lossless, threshold=0): %d\n", countNodes(root_lossless));
    printf("Quantidade de nos (lossy, threshold=%d): %d\n", threshold2, countNodes(root_lossy));
```

MODO LOSSY (THRESHOLD = 30)(CHAMA BUILDQUADTREE COM TOLERÂNCIA PARA COMPRIMIR A IMAGEM E CONTA OS NÓS)

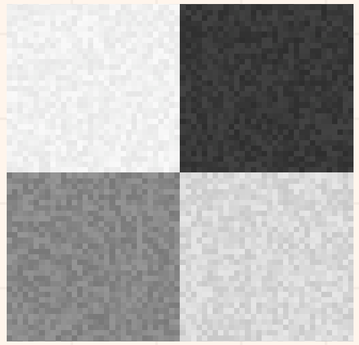
RESULTADOS E ANÁLISE

Comparação Visual e Estrutural

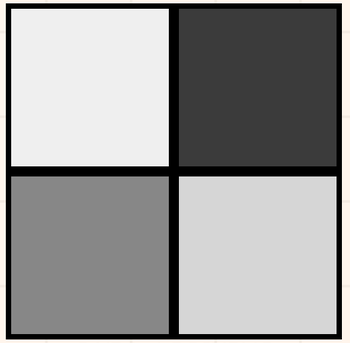
Representação	Elementos de Dados
Imagem Original (Array 64x64)	**4096** pixels
Quadtree Lossless (T=0)	**5461** nós
Quadtree Lossy (T=30)	5 nós



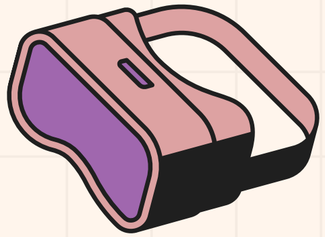
SAIDA_ORIGINAL.PNG

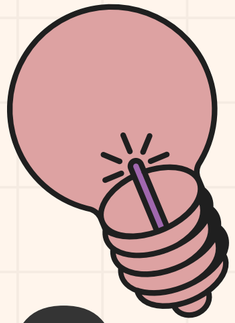


SAIDA_LOSSLESS.PNG



SAIDA_LOSSY.PNG





OBRIGADO PELA ATENÇÃO