

# UNIVERSIDAD NACIONAL DEL ALTIPLANO

**Facultad Ingeniería Mecánica, Eléctrica, Electrónica y Sistemas**

**Escuela Profesional de Ingeniería de Sistemas**



## **Trabajo 4 :**

**EJERCICIOS QUAD TREES**

**DOCENTE:**

**ING. COLLANQUI MARTINEZ FREDY**

**PRESENTADO POR:**

**CHOQUE CHURA OLIVER FRAN**

**SEXTO SEMESTRE 2024-I**

**PUNO - PERÚ**

**2024**

## LINK GITHUB

<https://github.com/OliverChoque/Quad-Trees.git>

## IMPLEMENTACIÓN EN JAVASCRIPT

### Ejercicio 1: Descomposición de Imágenes con Quad Trees

#### 1. Objetivo

Descomponer una imagen en escala de grises en una estructura de Quad Tree, identificando las regiones homogéneas según un umbral dado.

#### 2. Descripción del Problema

Dada una imagen en escala de grises, divide la imagen en cuadrantes recursivamente hasta que cada región sea homogénea según un umbral. Implementa una función que construya el Quad Tree correspondiente y otra función que permita visualizar la estructura del Quad Tree.

## CÓDIGO

```
<!DOCTYPE html>

<html lang="es">

<head>

  <meta charset="UTF-8">

  <title>Descomposición de Imágenes con Quad Trees</title>

  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">

  <style>

    body {

      display: flex;

      justify-content: center;

      align-items: center;

      height: 100vh;

      background-color: #f8f9fa; /* Color de fondo */

    }

    .container {

      text-align: center;

      max-width: 800px; /* Ancho máximo del contenido */

      padding: 20px;

    }

  </style>

</head>

<body>

  <div class="container">

    <h1>Descomposición de Imágenes con Quad Trees</h1>

  </div>

</body>

</html>
```

```

        .canvas-container {

            display: flex;

            justify-content: center;

            align-items: center;

        }

        canvas {

            border: 1px solid #ccc;

            display: block; /* Asegura que los canvas se muestren correctamente */

            margin: 10px; /* Agrega un pequeño margen */

        }

        input[type=range] {

            width: 80%;

            margin-top: 10px;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1 class="mb-4">Descomposición de Imágenes con Quad Trees</h1>

        <input type="file" id="imageInput" accept="image/*" class="mb-3" />

        <input type="range" id="thresholdSlider" min="0" max="255" value="10" class="form-range" />

        <p id="thresholdValue">Umbral: 10</p>

        <button id="buttonDecompress" class="btn btn-primary mt-3">Descomprimir</button>

        <div class="canvas-container mt-4">

            <canvas id="canvasOriginal" width="400" height="400"></canvas>

            <canvas id="canvasDecompressed" width="400" height="400"></canvas>

        </div>

    </div>

</body>

</html>

<script>

    document.getElementById('imageInput').addEventListener('change', function(event) {

        const file = event.target.files[0];

        const img = new Image();

        const canvasOriginal = document.getElementById('canvasOriginal');

        const ctxOriginal = canvasOriginal.getContext('2d');

        const canvasDecompressed = document.getElementById('canvasDecompressed');

        const ctxDecompressed = canvasDecompressed.getContext('2d');

    });

```

```

img.onload = function() {

    // Definir el tamaño máximo para los canvas

    const maxSize = 400;

    let scale = 1;

    if (img.width > img.height) {

        scale = maxSize / img.width;

    } else {

        scale = maxSize / img.height;

    }

    const newWidth = img.width * scale;

    const newHeight = img.height * scale;

    // Ajustar el tamaño de los canvas

    canvasOriginal.width = newWidth;

    canvasOriginal.height = newHeight;

    canvasDecompressed.width = newWidth;

    canvasDecompressed.height = newHeight;

    // Dibujar la imagen original en el canvas original

    ctxOriginal.drawImage(img, 0, 0, newWidth, newHeight);

    // Obtener los datos de escala de grises de la imagen

    const imageData = ctxOriginal.getImageData(0, 0, newWidth, newHeight);

    const grayData = getGrayScaleData(imageData);

};

img.src = URL.createObjectURL(file);

});

function getGrayScaleData(imageData) {

    const grayData = [];

    for (let y = 0; y < imageData.height; y++) {

        grayData[y] = [];

        for (let x = 0; x < imageData.width; x++) {

            const index = (y * imageData.width + x) * 4;

```

```

        const r = imageData.data[index];

        const g = imageData.data[index + 1];

        const b = imageData.data[index + 2];

        const gray = (r + g + b) / 3;

        grayData[y][x] = gray;

    }

}

return grayData;

}

class QuadTree {

    constructor(x, y, width, height, value = null) {

        this.x = x;

        this.y = y;

        this.width = width;

        this.height = height;

        this.value = value;

        this.children = [];

    }

    isLeaf() {

        return this.children.length === 0;

    }

}

function buildQuadTree(data, x, y, width, height, threshold, maxDepth = 5) {

    if (isHomogeneous(data, x, y, width, height, threshold)) {

        const avgValue = getAverageValue(data, x, y, width, height);

        return new QuadTree(x, y, width, height, avgValue);

    } else if (maxDepth === 0) {

        return new QuadTree(x, y, width, height, null);

    } else {

        const halfWidth = Math.floor(width / 2);

        const halfHeight = Math.floor(height / 2);

        const quadTree = new QuadTree(x, y, width, height);

        quadTree.children.push(buildQuadTree(data, x, y, halfWidth, halfHeight, threshold,
maxDepth - 1));
    }

}

```

```

        quadTree.children.push(buildQuadTree(data, x + halfWidth, y, halfWidth, halfHeight,
threshold, maxDepth - 1));

        quadTree.children.push(buildQuadTree(data, x, y + halfHeight, halfWidth, halfHeight,
threshold, maxDepth - 1));

        quadTree.children.push(buildQuadTree(data, x + halfWidth, y + halfHeight, halfWidth,
halfHeight, threshold, maxDepth - 1));

```

```

        return quadTree;

    }

}

```

```

function isHomogeneous(data, x, y, width, height, threshold) {

    let sum = 0;

    let sumSq = 0;

    let count = 0;

    for (let i = y; i < y + height; i++) {

        for (let j = x; j < x + width; j++) {

            const value = data[i][j];

            sum += value;

            sumSq += value * value;

            count++;

        }

    }

    const mean = sum / count;

    const variance = (sumSq / count) - (mean * mean);

    return variance <= threshold;

}

```

```

function getAverageValue(data, x, y, width, height) {

    let sum = 0;

    let count = 0;

    for (let i = y; i < y + height; i++) {

        for (let j = x; j < x + width; j++) {

            sum += data[i][j];

            count++;

        }

    }

```

```

    }

    return sum / count;
}

function visualizeQuadTree(quadTree, ctx, depth = 0) {
    if (quadTree.isLeaf()) {
        const grayValue = Math.round(quadTree.value);

        ctx.fillStyle = `rgb(${grayValue}, ${grayValue}, ${grayValue})`;

        ctx.fillRect(quadTree.x, quadTree.y, quadTree.width, quadTree.height);
    } else {
        ctx.strokeStyle = `rgba(0, 0, 0, 0.1)`;

        ctx.strokeRect(quadTree.x, quadTree.y, quadTree.width, quadTree.height);

        quadTree.children.forEach(child => visualizeQuadTree(child, ctx, depth + 1));
    }
}

document.getElementById('buttonDecompress').addEventListener('click', function() {
    const canvasOriginal = document.getElementById('canvasOriginal');

    const ctxOriginal = canvasOriginal.getContext('2d');

    const img = ctxOriginal.getImageData(0, 0, canvasOriginal.width, canvasOriginal.height);

    const grayData = getGrayScaleData(img);

    const threshold = parseInt(document.getElementById('thresholdSlider').value);

    const quadTree = buildQuadTree(grayData, 0, 0, canvasOriginal.width, canvasOriginal.height,
threshold);

    const canvasDecompressed = document.getElementById('canvasDecompressed');

    const ctxDecompressed = canvasDecompressed.getContext('2d');

    visualizeQuadTree(quadTree, ctxDecompressed);

    document.getElementById('thresholdValue').textContent = `Umbral: ${threshold}`;
});
</script>
</body>
</html>

```

## RESULTADO

Umbral: 69

### Descomposición de Imágenes con Quad Trees

Seleccionar archivo CR7.jfif

Umbral: 69

Descomprimir



Umbral: 170

### Descomposición de Imágenes con Quad Trees

Seleccionar archivo CR7.jfif

Umbral: 170

Descomprimir





## Ejercicio 2: Compresión de Imágenes con Quad Trees

### 3. Objetivo

Comprimir una imagen en escala de grises utilizando un Quad Tree y evaluar la calidad de la compresión.

### 4. Descripción del Problema

Dada una imagen en escala de grises, queremos representar la imagen utilizando un Quad Tree para reducir el tamaño de almacenamiento. Después, se debe reconstruir la imagen a partir del Quad Tree y comparar la imagen original con la imagen reconstruida.

## CÓDIGO

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Compresión de Imágenes con Quad Trees</title>

  <!-- Bootstrap CSS -->

  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">

  <style>

    body {

      display: flex;

      justify-content: center;

      align-items: center;

      height: 100vh;

      background-color: #f8f9fa;

    }

    .container {

      max-width: 800px;

      width: 100%;

      padding: 20px;

      background-color: #fff;

      box-shadow: 0 0 10px rgba(0,0,0,0.1);

    }

    h2 {

      text-align: center;
```

```

    }

    .btn-container {

        display: flex;

        justify-content: center; /* Centra los botones horizontalmente */

        margin-top: 20px;
    }

    .canvas-container {

        display: flex;

        justify-content: space-around; /* Separa los canvas */

        margin-top: 30px;

        text-align: center;

        padding: 10px 0; /* Añade espacio entre la sección de canvas y los botones */
    }

    .canvas-container > div {

        flex: 1; /* Hace que cada contenedor de canvas ocupe el mismo ancho */
    }

    canvas {

        max-width: 100%;

        height: auto;

        border: 1px solid #ddd;

        box-shadow: 0 0 5px rgba(0,0,0,0.1);
    }

    #stats {

        text-align: center;

        margin-top: 20px;
    }
}

</style>

</head>

<body>

    <div class="container">

        <h2 class="mt-4 mb-4">Compresión de Imágenes con Quad Trees</h2>

        <div class="form-group">

            <label for="upload">Selecciona una imagen:</label>

            <input type="file" id="upload" class="form-control-file" accept="image/*"/>

        </div>

        <div class="btn-container">

```

```

        <button id="compressButton" class="btn btn-primary" disabled>Comprimir</button>

        <button id="reconstructButton" class="btn btn-secondary" disabled>Reconstruir</button>
Reconstruida</button> <button id="downloadButton" class="btn btn-success" disabled>Descargar Imagen
    </div>

    <div class="canvas-container">
        <div>
            <h3>Original</h3>

            <canvas id="originalCanvas"></canvas>

        </div>

        <div>
            <h3>Reconstruida</h3>

            <canvas id="reconstructedCanvas"></canvas>

        </div>
    </div>

    <div id="stats" class="mt-4"></div>
</div>

<!-- Bootstrap JS and dependencies -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

<script>
    class QuadTreeNode {
        constructor(x, y, width, height, value) {
            this.x = x;
            this.y = y;
            this.width = width;
            this.height = height;
            this.value = value;
            this.children = [];
        }

        isLeaf() {
            return this.children.length === 0;
        }
    }

```

```

    }

    function buildQuadTree(data, width, height, x = 0, y = 0, currentWidth = width, currentHeight =
height, threshold = 10) {

        if (currentWidth <= 1 || currentHeight <= 1) {

            return new QuadTreeNode(x, y, currentWidth, currentHeight, data[y * width + x]);

        }

        const avg = averageValue(data, width, x, y, currentWidth, currentHeight);

        const diff = maxDifference(data, width, x, y, currentWidth, currentHeight, avg);

        if (diff < threshold) {

            return new QuadTreeNode(x, y, currentWidth, currentHeight, avg);

        }

        const halfWidth = Math.floor(currentWidth / 2);

        const halfHeight = Math.floor(currentHeight / 2);

        const node = new QuadTreeNode(x, y, currentWidth, currentHeight, avg);

        node.children.push(buildQuadTree(data, width, height, x, y, halfWidth, halfHeight,
threshold));

        node.children.push(buildQuadTree(data, width, height, x + halfWidth, y, halfWidth,
halfHeight, threshold));

        node.children.push(buildQuadTree(data, width, height, x, y + halfHeight, halfWidth,
halfHeight, threshold));

        node.children.push(buildQuadTree(data, width, height, x + halfWidth, y + halfHeight,
halfWidth, halfHeight, threshold));

        return node;

    }

    function averageValue(data, width, x, y, w, h) {

        let sum = 0;

        let count = 0;

        for (let i = y; i < y + h; i++) {

            for (let j = x; j < x + w; j++) {

                sum += data[i * width + j];

                count++;

            }

        }

        return sum / count;

    }

```

```

function maxDifference(data, width, x, y, w, h, avg) {

    let maxDiff = 0;

    for (let i = y; i < y + h; i++) {

        for (let j = x; j < x + w; j++) {

            maxDiff = Math.max(maxDiff, Math.abs(data[i * width + j] - avg));

        }

    }

    return maxDiff;

}

function reconstructImage(node, data, width, height) {

    if (node.isLeaf()) {

        for (let i = node.y; i < node.y + node.height; i++) {

            for (let j = node.x; j < node.x + node.width; j++) {

                data[i * width + j] = node.value;

            }

        }

    } else {

        node.children.forEach(child => reconstructImage(child, data, width, height));

    }

}

function calculateMSE(original, reconstructed) {

    let mse = 0;

    for (let i = 0; i < original.length; i++) {

        mse += Math.pow(original[i] - reconstructed[i], 2);

    }

    return mse / original.length;

}

function calculateStorageReduction(originalSize, compressedSize) {

    return ((originalSize - compressedSize) / originalSize) * 100;

}

function countNodes(node) {

    if (node.isLeaf()) {

        return 1;

    }

```

```

    }

    return 1 + node.children.reduce((sum, child) => sum + countNodes(child), 0);
}

function downloadImage(canvas) {

    const link = document.createElement('a');

    link.download = 'reconstructed_image.png';

    link.href = canvas.toDataURL('image/png');

    link.click();

}

let grayScaleData = [];

let imgWidth = 0;

let imgHeight = 0;

let quadTree;

document.getElementById('upload').addEventListener('change', function(e) {

    const file = e.target.files[0];

    const reader = new FileReader();

    reader.onload = function(event) {

        const img = new Image();

        img.onload = function() {

            const originalCanvas = document.getElementById('originalCanvas');

            const ctx = originalCanvas.getContext('2d');

            originalCanvas.width = img.width;

            originalCanvas.height = img.height;

            ctx.drawImage(img, 0, 0);

            const imgData = ctx.getImageData(0, 0, originalCanvas.width, originalCanvas.height);

            grayScaleData = [];

            for (let i = 0; i < imgData.data.length; i += 4) {

                const gray = 0.299 * imgData.data[i] + 0.587 * imgData.data[i + 1] + 0.114 *
imgData.data[i + 2];

                grayScaleData.push(gray);

            }

            imgWidth = img.width;

            imgHeight = img.height;

            document.getElementById('compressButton').disabled = false;

        };

        img.src = event.target.result;

```

```

    };

    reader.readAsDataURL(file);
  });

document.getElementById('compressButton').addEventListener('click', function() {

  quadTree = buildQuadTree(grayScaleData, imgWidth, imgHeight);

  const originalSize = grayScaleData.length;

  const compressedSize = countNodes(quadTree);

  const reductionPercentage = calculateStorageReduction(originalSize, compressedSize);

  document.getElementById('stats').innerHTML = `

    <p>Reducción en el almacenamiento: ${reductionPercentage.toFixed(2)}%</p>

  `;

  document.getElementById('reconstructButton').disabled = false;
});

document.getElementById('reconstructButton').addEventListener('click', function() {

  const reconstructedData = new Array(grayScaleData.length).fill(0);

  reconstructImage(quadTree, reconstructedData, imgWidth, imgHeight);

  const mse = calculateMSE(grayScaleData, reconstructedData);

  displayReconstructedImage(reconstructedData, imgWidth, imgHeight);

  document.getElementById('stats').innerHTML += `

    <p>Error Cuadrático Medio (MSE): ${mse.toFixed(2)}</p>

  `;

  document.getElementById('downloadButton').disabled = false;
});

document.getElementById('downloadButton').addEventListener('click', function() {

  const reconstructedCanvas = document.getElementById('reconstructedCanvas');

  downloadImage(reconstructedCanvas);
});

function displayReconstructedImage(data, width, height) {

  const reconstructedCanvas = document.getElementById('reconstructedCanvas');

  reconstructedCanvas.width = width;

  reconstructedCanvas.height = height;

  const ctx = reconstructedCanvas.getContext('2d');

  const imgData = ctx.createImageData(width, height);

```

```
for (let i = 0; i < data.length; i++) {  
  
    imgData.data[i * 4] = data[i];  
  
    imgData.data[i * 4 + 1] = data[i];  
  
    imgData.data[i * 4 + 2] = data[i];  
  
    imgData.data[i * 4 + 3] = 255; // alpha  
  
}  
  
ctx.putImageData(imgData, 0, 0);  
  
}  
  
</script>  
  
</body>  
  
</html>
```

## RESULTADO

### Compresión de Imágenes con Quad Trees

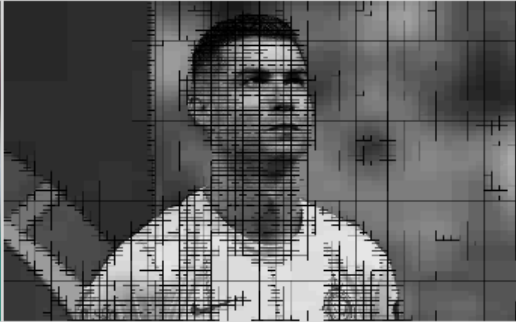
Selecciona una imagen:

CR7.jfif

ComprimirReconstruirDescargar Imagen Reconstruida

Original

Reconstruida



Reducción en el almacenamiento: 93.84%

Error Cuadrático Medio (MSE): 2584.84