Ramdeobaba University,
Nagpur

Department of Computer Science and Engineering

Session: 2025-26

Subject: Design and Analysis of Algorithms (DAA) Lab
Project IIIrd semester

LAB PROJECT REPORT

Topic: Image Compression using Divide and Conquer Strategy (Region Quadtree)

Field	Detail
Group Members (Section-Batch-Rollno.)	Sujal Singh (A4-B4-53) Kishan Saini (A4-B4-55)
GitHub Link	https://github.com/SujalSingh248/DAA_LAB Project.git
Website(Deployment Link)	https://imagecompresser25.netlify.app/

1. Objectives

The primary goals of this project were to achieve efficient image compression through structural and algorithmic implementation:

1. **Data Structure Implementation:** Implement the **Region Quadtree** structure to efficiently partition a two-dimensional image based on color similarity.
2. **Algorithmic Development:** Develop a **recursive splitting algorithm** to divide image regions based on a color variance threshold (ϵ).
3. **Complexity Analysis:** Analyze the **time and space complexity** of the Quadtree construction and image reconstruction (decompression) to assess algorithmic efficiency.
4. **Performance Evaluation:** Evaluate the **compression ratio** achieved and analyze the trade-off between the chosen threshold and the resulting visual quality.

2. Introduction

Digital images consume significant memory and bandwidth. **Image Compression** is vital for reducing the number of bits required to store or transmit an image.

Our solution employs the **Quadtree**, a hierarchical data structure excellent for partitioning 2D space. Applied to image compression, the Quadtree works on the **Divide and Conquer** principle:

- The entire image is the **root node**.
- Any region deemed **uniform** (colors are very similar) becomes a **leaf node**, storing only the average color.
- Any region deemed **non-uniform** (high color variance) is recursively split into four smaller **quadrants** (NW, NE, SW, SE).

This adaptive process results in **lossy compression**, where large, smooth areas are heavily compressed, while intricate details are preserved using smaller blocks.

3. Algorithms/Technique Used

The project is driven by the **Region Quadtree Decomposition Algorithm**.

Quadtree Node Structure

Each node in the tree conceptually stores:

- The bounding coordinates of its image region.
- Pointers to its four children (NW, NE, SW, SE).
- If it is a **Leaf Node**, it stores the average color value (RGB).

Algorithm: Quadtree Construction (Compression)

The algorithm proceeds recursively, checking for uniformity at each step:

1. **Base Case Check:** If the region is a single pixel, it terminates and stores the pixel's color. 2. **Variance Calculation:** Calculate the color variance (Max $\text{value} - \text{Min value}$) for all three RGB channels within the current region.
3. **Splitting Decision:**
    - **Compress (Leaf Node):** If the color variance for **all channels** is less than or equal to the threshold (epsilon), the recursion stops. The current node becomes a leaf node, storing the region's average color.
    - **Split (Internal Node):** If the variance exceeds ϵ, the region is non-uniform. The current node becomes an internal node.
4. **Recursion (Divide):** The region is divided into four sub-quadrants, and the construction algorithm is called recursively on each quadrant.

Divide and Conquer Strategy

The Quadtree structure perfectly embodies the Divide and Conquer strategy:

- **Divide:** A non-uniform problem space (image region) is split into four smaller, independent sub problems (quadrants).
- **Conquer:** The compression logic is applied recursively to each quadrant until the base case (uniformity or minimum size) is met.
- **Combine:** The internal nodes naturally combine the results by maintaining pointers to their compressed sub-regions, forming the final tree structure.

4. Time Complexity and its Explanation

The efficiency of the Quadtree is analyzed using the Master Theorem.

Metric	Analysis	Complexity
Recurrence Relation	**T(N) = 4T(N/4) + f(N)**, where N is the number of pixels. The function f(N) is the cost of calculating uniformity, which is O(Block Size).	
Worst Case (Construction)	The image is highly complex, forcing decomposition down	**O(N)**

	to single pixels (high cost of f(N)).	
Best Case (Construction)	The image is a solid color, stopping recursion at the root node.	**O(1)**
Image Reconstruction (Decompression)	Requires traversing all nodes (internal and leaf) to fill the output array.	**O(N)**

The overall time complexity for the Quadtree compression and decompression is **O(N)**, confirming its efficiency for processing large image data sets.

5. Results

The performance was evaluated by comparing the number of nodes (data points) required to represent the original image versus the compressed image at a specified threshold.

Comparison Analysis:

Image / Threshold	Number of Data Points	Compression Ratio
Original	6,562,500 pixel-values	1.00
Compressed	838,224 nodes	≈ 7.83:1

Discussion: The results show a significant compression ratio of **7.83:1** at a threshold (epsilon) of 100. This is achieved because the Quadtree algorithm is **adaptive**; it uses large, low-detail blocks for uniform areas (like backgrounds) and allocates smaller blocks (deeper recursion) to preserve critical visual features (like edges and fine details). The choice of epsilon directly dictates this trade-off between file size and image fidelity.

6. Conclusion and Future Scope

Conclusion

We successfully demonstrated the application of the **Divide and Conquer** paradigm through the **Region Quadtree** for efficient image compression. The technique's main strength is its ability to use variable block sizes, ensuring high compression in smooth areas while maintaining detail where the color variance is high. The algorithm's confirmed **O(N) time complexity** validates its performance for scalable image processing tasks.

Future Scope

We propose the following avenues for extending this project:

1.Parallelization: Utilize CUDA or MPI to parallelize the independent recursive calls to the four quadrants, potentially achieving faster compression times for large, high-resolution images.

2.Adaptive Thresholding:Implement a non-uniform threshold based on localized image features (e.g., smaller \epsilon near detected edges) to dynamically optimize quality in crucial areas.

3. Higher Dimensions:Extend the concept to Octrees for compression or partitioning of 3D data, such as video volumes or medical imaging scans.