IMAGE SEGMENTATION

ID	الاسم
2022170099	بولس عوني لطيف يسي مرزوق
2022170111	جلال محمد جلال محمد طعيمة
2022170475	نورين محمد حمدين عبدالمقصود
2022170108	توماس سيدهم بسطوروس دميان
2022170232	عبدالرحمن حسين محمد عبدالله

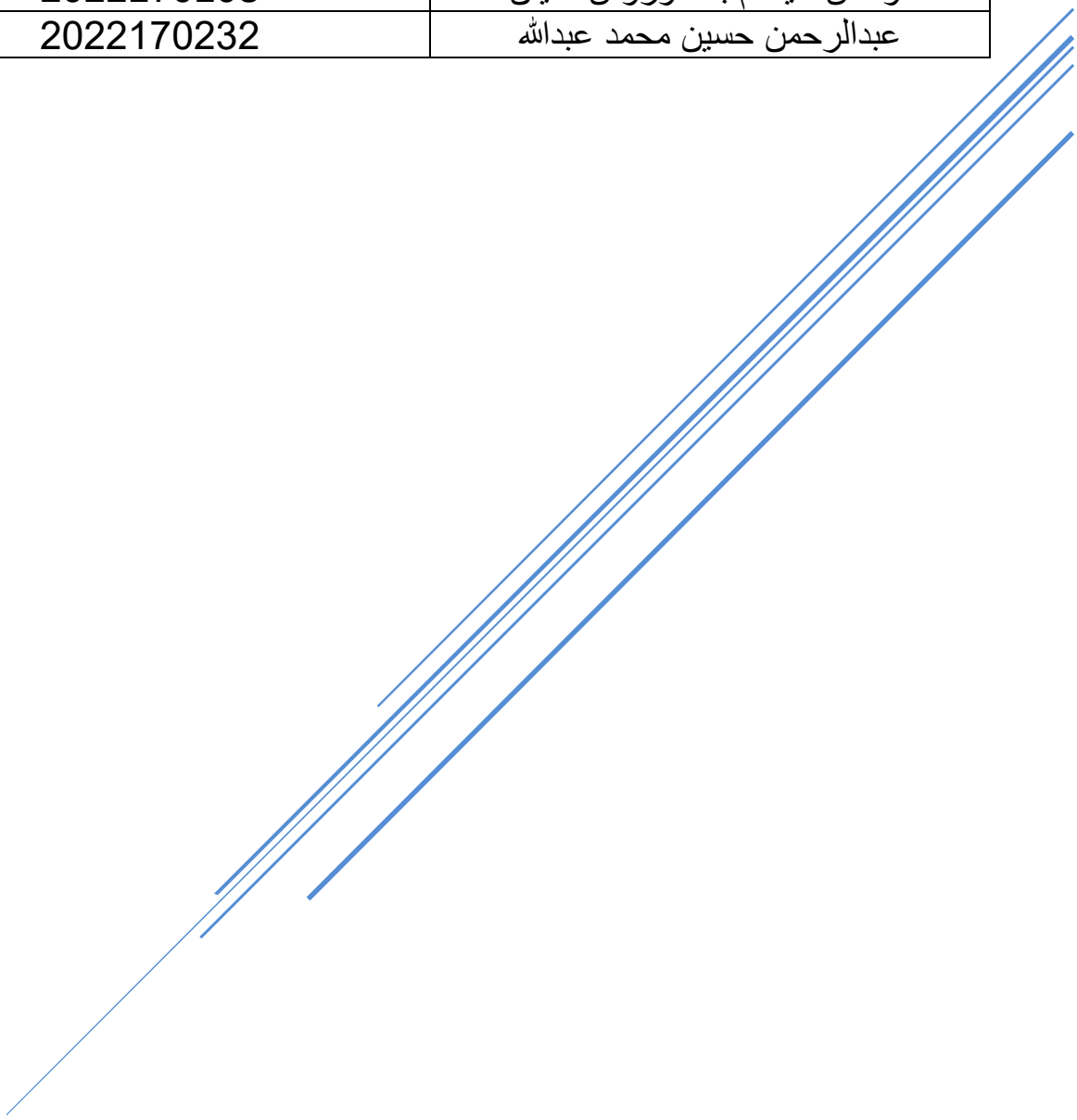

Image Segmentation Documentation

Overview

The **Image Segmentation Project** is designed to perform image segmentation by partitioning an input image into distinct regions based on pixel similarity across the RGB color channels. The approach leverages graph-based segmentation with the Disjoint Set Union (DSU) data structure to identify connected components in graphs constructed from the image pixels.

This project segments an image by constructing three separate weighted graphs—one for each color channel (Red, Green, Blue)—where nodes represent pixels, edges represent neighborhood pixel connections, and edge weights correspond to the difference in pixel intensity. The final segmentation result is derived by intersecting the three segmentations (one per channel) to produce a consolidated segmentation map highlighting regions consistent across all three channels.

Key Concepts

Graph Construction

- Each pixel in the image is treated as a graph node.
- Edges are formed between each pixel and its 8 neighbors (surrounding pixels in a 3x3 window).
- Edge weights are the absolute difference in intensity values of the respective color channel (red, green, or blue) between connected pixels.
- This process creates three separate graphs representing the Red, Green, and Blue channels.

Image Segmentation Using Disjoint Set Union (DSU)

- The graphs are sorted by edge weight using Counting Sort, an efficient O(M) sorting algorithm suited for edge weights bounded by 0–255.
- The DSU (Disjoint Set Union) structure groups nodes into connected components by uniting nodes connected by edges with low weight (similar intensity).
- The union operation also considers a threshold to decide if regions should be merged or kept separate based on edge weight and component size (following the methodology of graph-based image segmentation).
- This produces three separate segmentations—one per color channel.

Intersection of Segmentations

- The segmentation results from the three channels are intersected to find regions where pixels belong to the same component in all three segmentations.
- This intersection creates the final segmentation labeling.
- The intersection is implemented in parallel to improve performance over large images.

Region Color Assignment and Output

- Each segment (connected component) is assigned a random RGB color to facilitate visual distinction.
- The original image matrix is updated with these new colors.
- Segment size statistics (number of pixels per segment) are recorded.
- An optional output log file records the total number of segments and their sizes for further analysis.

Code Workflow

1. **Graph Construction (GraphConstruct)**
    - Iterates over every pixel in the image.
    - For each pixel, checks all 8 neighbors.
    - Adds edges with weights representing pixel intensity differences in a specified color channel.
2. **Edge Sorting (CountingSortEdges)**
    - Performs counting sort on edges by weight for efficient sorting within fixed range [0..255].
3. **Segmentation per Color Channel (ImageSegmentation)**
    - Sorts edges and uses DSU to merge connected components with low difference.
    - Runs union operations on edges to form segments.
4. **Parallel Graph Construction (ConstructGraphs)**
    - Builds the three color channel graphs in parallel tasks.
5. **Parallel Segmentation (SegmentGraphs)**
    - Segments the Red and Green graphs in parallel, segments Blue graph sequentially.
6. **Intersection of Segmentations**
    - Creates a new DSU structure to combine segmentation results from all three channels.
    - Merges nodes that belong to the same component across all three color graphs.
    - Performed in parallel for efficiency.

7. **Assign Colors to Segments**
    - Randomly assigns a distinct color per connected component.
    - Updates the image matrix with the segment colors.
    - Tracks and sorts segment sizes.
8. **Logging**
    - Optionally writes segment counts and sizes to a file.
9. **Manual Segment Merging**
    - Allows manual merging of multiple segments with validation and image update.

Time Complexity Analysis

Step	Complexity	Explanation
Graph Construction	O(N * M * 8) ≈ O(N*M)	Each pixel checks up to 8 neighbors (constant). N and M are image dimensions.
Edge Sorting (Counting Sort)	O(E + W) ≈ O(N*M)	Counting sort is linear; E = edges, W = max weight (255). Edges ~ 8*N*M.
Disjoint Set Union Operations	O(E * α(N*M))	α is inverse Ackermann function, practically constant, E edges processed.
Intersection (Parallel)	O(N * M)	Every pixel checked against neighbors in parallel.
Color Assignment & Mapping	O(N * M)	Assign colors and count pixels per segment.

- Overall, the algorithm is primarily linear in the number of pixels (N*M), with some logarithmic factors due to union-find.

- The use of parallelism on independent tasks (graph construction, segmentation, intersection) greatly improves wall-clock time on multicore CPUs.

- Memory complexity is O(N*M) for storing pixel data, edges, and DSU structures.