Image Color Quantization

[Sample Input/Output 3](#_Toc25485227)

[Problem Definition 3](#_Toc25485228)

[What is Image Quantization? 3](#_Toc25485229)

[Some Usages 4](#_Toc25485230)

[Main Steps 5](#_Toc25485231)

[Related Image Terminologies 5](#_Toc25485232)

[Descriptions and Suggestions 8](#_Toc25485233)

[Clustering 8](#_Toc25485234)

[Single-linkage Clustering 9](#_Toc25485235)

[Image Quantization Algorithm 10](#_Toc25485236)

[Project Requirements 12](#_Toc25485237)

[Required Implementation 12](#_Toc25485238)

[Input 13](#_Toc25485239)

[Output 13](#_Toc25485240)

[Test Cases 13](#_Toc25485241)

[Deliverables 14](#_Toc25485242)

[Implementation (60%) 14](#_Toc25485243)

[Document (40%) 14](#_Toc25485244)

[Allowed Codes [if any] 14](#_Toc25485245)

[Milestones 14](#_Toc25485246)

[BONUSES 15](#_Toc25485247)

[FIRST: Automatic Detection of the Number of Clusters 15](#_Toc25485248)

[SECOND: Better Way for Finding the (K) Clusters 15](#_Toc25485249)

[Appendix: Template Code Description 16](#_Toc25485250)

# Sample Input/Output



**Input Image**



**Output Image (2 clusters)**

**SEGMENTATION**

**COMPRESSION**

**Input Image**

1600×1000

**24 bit/pixel** (~4.6 MB)



**Output Image**

1600×1000

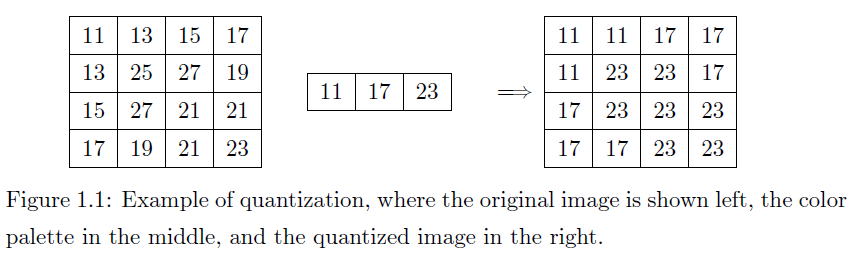
**12 bit/pixel** (~2.3 MB)

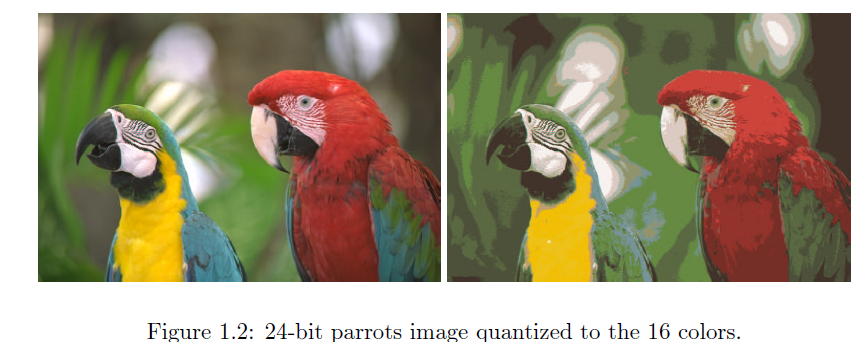
# Problem Definition

## What is Image Quantization?

The idea of *color quantization* is to reduce the number of colors in a full resolution digital color image (24 bits per pixel) to a smaller set of representative colors called ***color palette*.** Reduction should be performed so that the quantized image differs as little as possible from the original image. Algorithmic optimization task is to find such a color palette that the overall distortion is minimized.

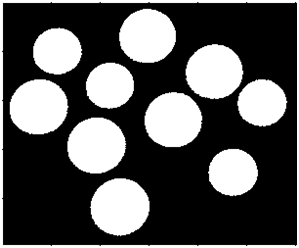
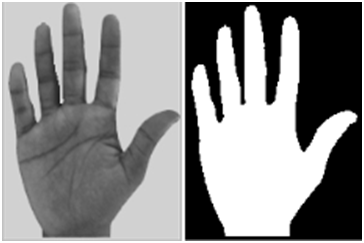
An example of color quantization is depicted in the following Figure. First, a color palette is found by using clustering algorithm and then the original image values are replaced by their closest values in the palette.



****

## Some Usages

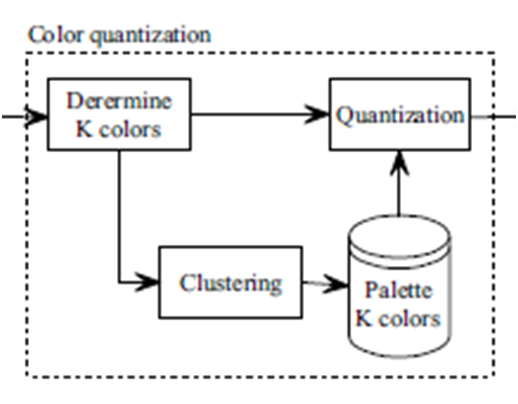
1. ***Target different devices:*** color quantization is critical for displaying images with many colors on devices that can only display a limited number of colors, usually due to memory limitations.
2. ***Image compression:*** by reducing number of bits per pixels without affecting the image view. It’s used as a step in the compression pipeline of most common formats like JPEG and MPEG.
3. ***Image segmentation:*** is the process extracting useful objects from an image. It usually done by assigning a label to every pixel in an image such that pixels with the same label share certain characteristics (e.g. same colors). Examples are shown in the figure below:



## Main Steps

Color quantization consists of two main steps:

1. ***Palette Generation:*** A palette generation algorithm finds a smaller representative set of colors ***C***= *{c1,c2,c3,…,ck}*  from the *D* distinct colors.
2. ***Quantization:*** by mapping the original colors to the palette colors.



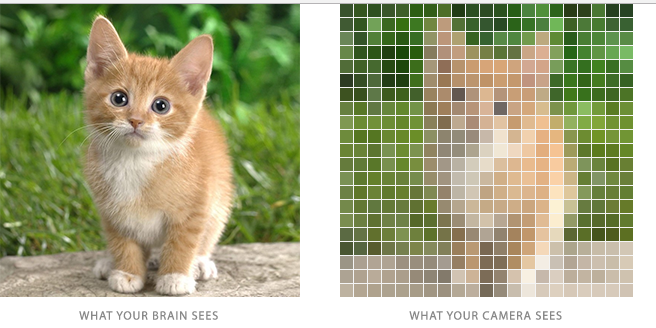
**Sample Input**

## Related Image Terminologies

### Digital Image

It’s an electronic snapshot taken of a scene or scanned from documents, such as photographs, manuscripts, printed texts, and artwork.

It’s made of picture elements called pixels. Typically, pixels are organized in an ordered rectangular array. Each pixel has its own intensity value, or brightness which is represented in binary code (zeros and ones).



### Color depth

Also known as **bit depth**, is the number of bits used to indicate the color of a single pixel.

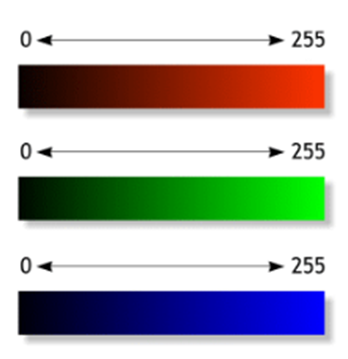
The size of an image is determined by two factors:

1. Dimensions: width and height of the 2D pixel array.
2. Color depth: number of bits per pixels

### Intensity

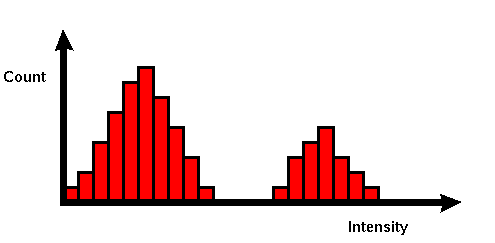
The intensity of a pixel is expressed within a given range between a minimum and a maximum value [Inclusive], based on the color depth of the pixel.

True Color images have intensity from the darkest (0) and lightest (255) of three different color channels, **R**ed, **G**reen, and **B**lue. Each channel has a range from 0 to 255 as shown in Figure below. So we need 8+8+8=24 bits to represent 1 pixel color which means we have 224 = 16,777,216 different colors.

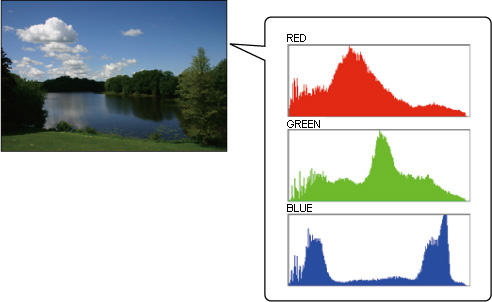


### Histogram

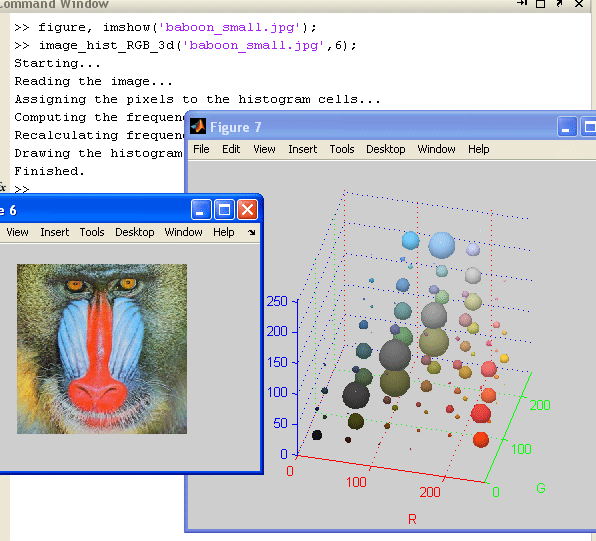
An **image histogram** is a graph showing the number of pixels in an image at each different intensity value found in that image.



Usually, there’s a different histogram for each channel (**R**ed, **G**reen, and **B**lue). Since each channel is represented by 8 bits, so, there are 256 possible values on x-axis. The histogram will graphically display 256 numbers showing the distribution of pixels amongst those values.



Another way is to construct a **3D histogram**, with the three axes representing the intensity values of red, blue and green channels (each one ranged from 0 to 255). The histogram value at certain point represents the number of pixels that have the RGB value at this point.

****

# Descriptions and Suggestions

Most standard techniques treat color quantization as a problem of clustering points in three-dimensional space, where the points represent colors found in the original image and the three axes represent the three color channels.

## Clustering

### Definition

**Clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense or another) to each other than to those in other groups (clusters).

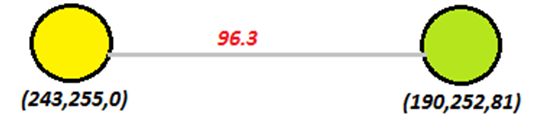
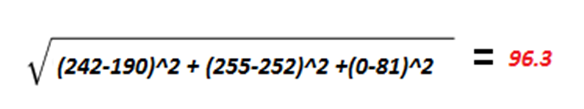
From the definition we can infer that:

1. **Set of objects** 🡪 are the set of distinct colors in the image ( each color is a point in 3d (RGB) space)
2. **Similarity measure** is required for grouping similar set of point colors together 🡪 the Euclidean Distance between RGB intensity values of the color points can be used here.

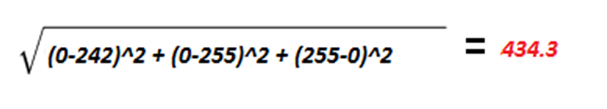
The Euclidean Distance between TWO colors is defined as:



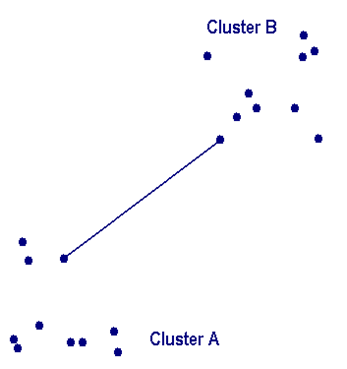
**Example1:** two similar colors with their Euclidean Distance (**small**)



**Example2:** two different colors with their Euclidean Distance (**large**)



1. **Grouping of points** with minimal Distances between them in one cluster 🡪 means (is equivalent to) producing clusters with maximal spacing.



**Maximal Spacing**

1. We can assume the number of groups/clusters.

### Objective

Given a **distance measure** and a **desired number of clusters** 🡺 produce **K clusters** with maximal Spacing which means grouping distinct points with minimal distances into one cluster.

## Single-linkage Clustering

### Definition

It is one of several methods of [hierarchical clustering](https://en.wikipedia.org/wiki/Hierarchical_clustering). It is based on grouping clusters in bottom-up fashion (agglomerative clustering). At each step, it combines two clusters that contain the closest pair of elements not yet belonging to the same cluster as each other.

With single linkage method (also called nearest neighbor method), the distance between two clusters is the minimum distance between an observation in one cluster and an observation in the other cluster which is defined as the Euclidean Distance.

### General Algorithm

The Greedy strategy of Single-linkage clustering:

1. Begin with the disjoint clusters: each point is considered a separate cluster.
2. Find the most similar pair of clusters in the current state, say pair (r), (s), according to *d*[(*r*),(*s*)] = min *d*[(*i*),(*j*)] where the minimum is over all pairs of clusters in the current state.
3. Merge clusters (*r*) and (*s*) into a single cluster to form the next state.
4. Repeat steps (2) to (4) until obtaining the desired number of clusters (K).
5. Find a representative point for each cluster. Usually by averaging all points in each cluster (called: centroid of the cluster ci).
6. Color Palette contains the centroids/representatives of the K clusters
7. Replace all points in each cluster by its representative point… Which means we need Log2(K) bits to represent the K clusters

## Image Quantization Algorithm

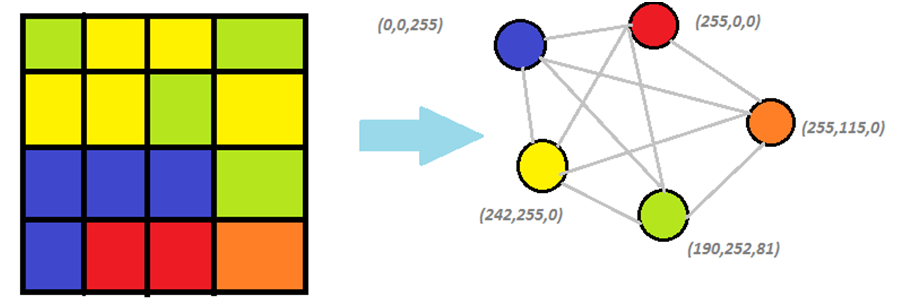
To Apply the Single-linkage Clustering algorithm on the Image Quantization Problem, we need to

1. Find the **distinct colors** *D = {d1, d2, d3 ….dm}* from theinput image. Can be known from the image histogram.
2. Construct a **fully-connected undirected weighted graph** *G* with

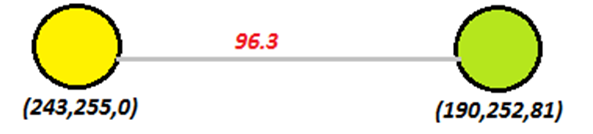
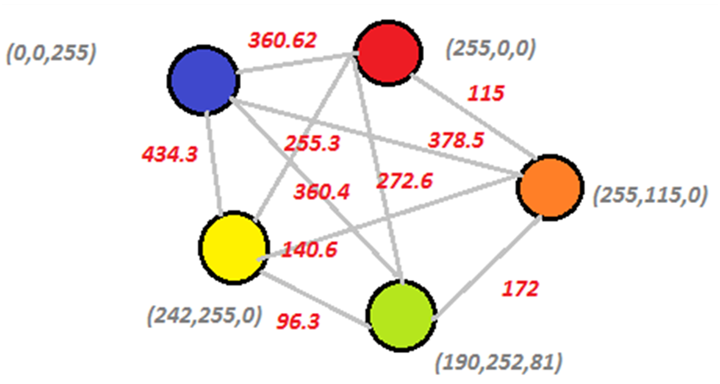
* *D* vertices (number of distinct colors).
* Each pair of vertices is connected by a single edge.
* Edge weight is set as the Euclidean Distance between the RGB values of the 2 vertices.

**Example:**

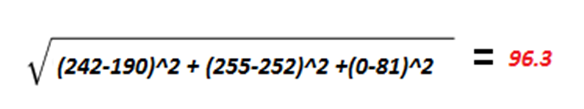
**4 x 4 image grid**



**Corresponding Graph**

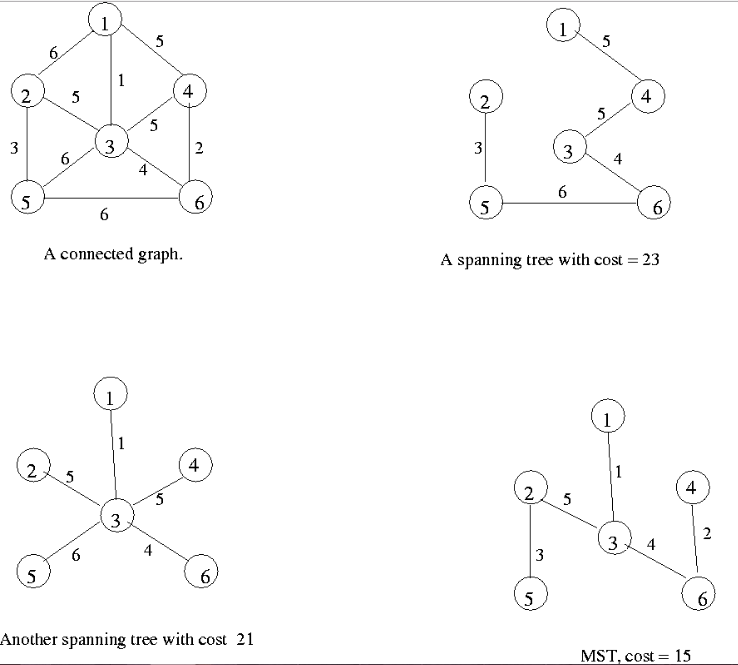


**Example of one edge**

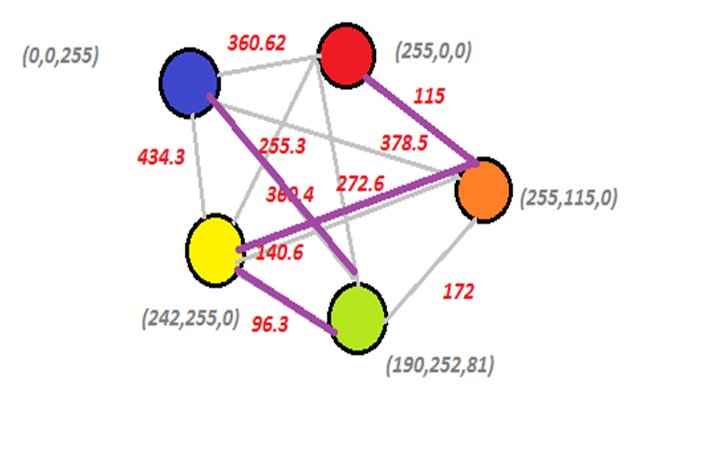


1. Construct  a [minimum-spanning-tree algorithm](https://en.wikipedia.org/wiki/Minimum_spanning_tree#Algorithms) (a [greedy algorithm](https://en.wikipedia.org/wiki/Greedy_algorithm) in [graph theory](https://en.wikipedia.org/wiki/Graph_theory))

* **Input:** connected undirected weighted graph
* **Output:** a tree that keeps the graph connected with minimum total cost
* **Methodology:** treats the graph as a forest and each node is initially represented as a tree. A tree is connected to another only and only if it has the least cost among all available.
* **Conclusion:** the Minimum Spanning Tree is an implementation of single linkage clustering Strategy that repeats merging distinct points with minimal distances into a single cluster

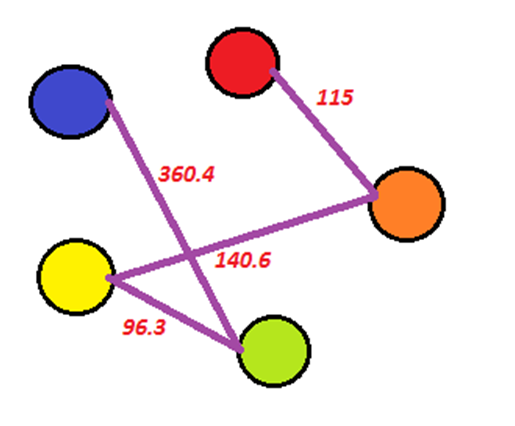


So the Minimum Spanning tree of the Distinct Color Graph will be:



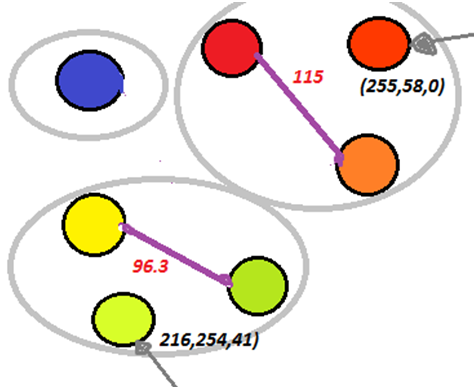
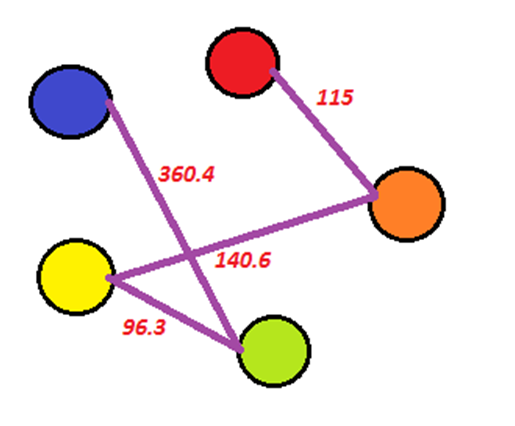
**Input Graph**

**Corresponding MST**



1. Extract the desired number of clusters (K) with maximum distances between each other.
2. Find the representative color of each cluster.
3. Quantize the image by replacing the colors of each cluster by its representative color.

**MST**



**Clusters (K = 3)**

**Representative color1:**

Average of the RGB of the 2 colors

R=(255+255)/2

G=(0+155)/2

B=(0+0)/2

**Representative color2:**

Average of the RGB of the 2 colors

R=(242+190)/2

G=(255+252)/2

B=(0+81)/2

# Project Requirements

## Required Implementation

|  |  |  |
| --- | --- | --- |
| **Requirement** | | **Performance** |
| 1. Construct the graph by and calculating distances between them. | 1. finding distinct colors | **Space:** **bounded by** **D2**, D is the number of distinct colors in the image  **Time:** **bounded by** **O(N2)**, N is the image width (or height) |
| 1. calculating distances between them. | **Time:** **bounded by** **O(D2)**, D is the number of distinct colors in the image |
| 1. Find the minimum spanning tree from the graph. | | **Time:** should be **proportional to** **O(E Log(V))**, E is number of edges, V is number of vertices |
| 1. Extract the **K** clusters from the minimum spanning tree with maximal spacing between them. | | **Time:** should be **bounded by O(K×D)**, D is the number of distinct colors |
| 1. Find the representative color of each cluster. | | **Time:** should be **bounded by O(D)**, D is the number of distinct colors |
| 1. Quantize the image by replacing the colors of each cluster by its representative color. | | **Time:** **bounded by** **O(N2)**, N is the image width (or height) |

## Input

1. Image (2D array of pixels)
2. Desired number of clusters (K)

## Output

1. Quantized image
2. Color palette

## Test Cases

### SAMPLE TESTS

* The algorithm can be tested on any true color picture on the computer.
* You have **5 Sample test cases with their results** that can be used for debugging/tracing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Image Name** | **# Distinct Colors** | **MST Sum** | **K** | **Result Image** |
| Sample.Case1 | 5 | 730.6 | 3 | Sample.Case1\_K=3 |
| Sample.Case2 | 3 | 322.6 | 2 | Sample.Case2\_K=2 |
| Sample.Case3 | 2266 | 6106.9 | 500 | Sample.Case3\_K=500 |
| Sample.Case4 | 69 | 1120.7 | 10 | Sample.Case4\_K=10 |
| Sample.Case5 | 256 | 441.7 | 32 | Sample.Case4\_K=32 |

### COMPLETE TESTS

* You have 3 level of complete test cases according to the image sizes, together with their results:

1. **Small:** image dimension is bounded by **O(1000×1000).**
2. **Medium:** image dimension is bounded by **O(5,000×5,000).**
3. **Large:** image dimension is bounded by **O(10,000×10,000).**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Image Name** | **# Distinct Colors** | **MST Sum** | **K** | **UPPER LIMIT TIME** | **Result Image** |
| Small.Case1 | 8,708 | 11,785.1 | 192 | 1 min | Small.Case1\_K=192 |
| Small.Case2 | 10,265 | 19,888.8 | 2160 | 1 min | Small.Case2\_K=2160 |
| Medium.Case1 | 27,410 | 40,616.4 | 1737 | 3 min | Medium.Case1\_K=1737 |
| Medium.Case2 | 20,041 | 44,831.7 | **2284** | 3 min | Medium.Case2\_K=2284 |
| Large.Case1 | 56,328 | 118,145.1 | **3829** | 6 min | Large.Case1\_K=3839 |
| Large.Case2 | 54,223 | 80,957.2 | 25,666 | 6 min | Large.Case2\_K=25666 |

# Deliverables

## Implementation (60%)

1. Graph construction.
2. EFFICIENT implementation of minimum spanning tree.
3. Extracting the K clusters.
4. Palette generation by calculating the clusters centroids.
5. Mapping the original colors to the palette colors.

## Document (40%)

1. Graph construction description and code.
2. Minimum spanning tree code.
3. Palette generation code.
4. Detailed analysis of the above codes.

## Allowed Codes [if any]

* Given template to open and display the images.

Refer to [**Appendix: Template Code Description**](#_Appendix:_Template_Code) for more details.

* Priority Queue Code [**MUST UNDERSTAND ITS CODE AND ANALYZE CORRECTLY**]

# Milestones

|  |  |  |
| --- | --- | --- |
|  | **Deliverables** | **Due to** |
| **Milestone1** | 1. Construct a weighted graph for distinct colors in the image 2. Implementing the minimum spanning tree of the graph. 3. Extracting the K clusters from the minimum spanning tree. 4. Calculating their centroids (Palette Generation) 5. Quantize the image by replacing the colors of each cluster by its representative color. 6. Documentation | **(START of Week Before Practical Exam)** |
| **For Milestone1:**   * + **MUST** deliver the required tasks and **ENSURE** it’s worked correctly   + **MUST** deliver in your scheduled time (TO BE ANNOUNCED) | | |

# BONUSES

## FIRST: Automatic Detection of the Number of Clusters

Clustering the distinct color points into disjoint groups **without** knowing the predefined number of clusters using the following criteria:

1. Computes the standard deviation of the edges in minimum spanning tree and store it
2. Choose an edge that leads to max standard deviation reduction once it is removed to obtain a set of two disjoint sub-trees.
3. Compute standard deviation reduction in current iteration.
4. Repeat steps (2) and (3) until convergence 🡪 the difference between standard deviation reduction in current iteration and previous iteration is very small < 0.0001

## SECOND: Better Way for Finding the (K) Clusters

**Repeat**

{// This Loop is entered with the Constructed Graph of Distinct Colors

1. Compute the Minimum Spanning Tree of the set of points/colors in the graph
2. Compute the Mean and STDdev of the edges in the MST
3. For each edge in the MST > Mean+STDdev, remove it to obtain a set of disjoint sub trees (clusters).
4. Check:
5. If the obtained number of clusters (Q) is less than the desired one (K):
6. Remove edges that have highest weights until reaching the desired number of clusters.
7. Return the K clusters and compute their centroids.
8. Else, if the obtained number of clusters (Q) is greater than the desired one (K):
9. Create a new empty set (S) of points
10. For each cluster q in Q
11. Compute its centroid
12. Find a representative point: a point in the cluster itself that has minimum distance to the centroid
13. Add this representative point in the new set of points (S)
14. Now the new set of points (S) has a less number of points than the original graph
15. Construct a graph from this new set of points (S)

**Until Q=K**

Return K clusters

}

# Appendix: Template Code Description

C# Code contains **ImageOperations** class with the following functionalities:

1. Open image & load it in a 2D array stored in a global variable of type MyImage[[1]](#footnote-1) [,]called ImageMatrix

RGBPixel [,] OpenImage(string ImagePath)

1. Get width and height of the image matrix

int GetHeight(RGBPixel[,] ImageMatrix)

int GetWidth(RGBPixel[,] ImageMatrix)

1. Display an image on a given PictureBox control

void DisplayImage(RGBPixel[,] ImageMatrix,PictureBox PicBox)

1. RGBPixel is a structure defined in the code to hold the Red, Green, Blue values of each pixel [↑](#footnote-ref-1)