Image Quantization Analysis

T055

# GetDistinctColorsList(ImageMatrix):

private static List<int> GetDistinctColorsList(RGBPixel[,] ImageMatrix)

{

HashSet<int> distinctColors = new HashSet<int>();

foreach (RGBPixel pixel in ImageMatrix)

distinctColors.Add(RGBPixel.Hash(pixel)); // O(N^2) if C# handles resizeing well

// Can we do better?

List<int> colorsList = distinctColors.ToList();

colorIndices = new Dictionary<int, int>(distinctColors.Count);

for (int i = 0; i < colorsList.Count; i++) // O(D)

colorIndices.Add(colorsList[i], i);

return colorsList;

}

Final Order: O(N^2)

# ReduceImageColors(ImageMatrix, ColorPallette, ClusterIndices):

// EXPECTED TO CHANGE

private static void ReduceImageColors(RGBPixel[,] ImageMatrix, Dictionary<int, RGBPixel> ColorPallette, Dictionary<int, short> clusterIndices)

{

int rows = ImageOperations.GetHeight(ImageMatrix);

int columns = ImageOperations.GetWidth(ImageMatrix);

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < columns; j++)

{

RGBPixel currentColor = ImageMatrix[i, j];

int currentColorIndex = colorIndices[RGBPixel.Hash(currentColor)];

int currentColorClusterIndex = clusterIndices[currentColorIndex];

RGBPixel newColor = ColorPallette[currentColorClusterIndex];

ImageMatrix[i, j] = newColor;

}

}

}

Final Order: O(N^2)

# Bonus 1 (Finding K using MSDR):

// EXPECTED TO CHANGE.

GetColorPallette(clusters):

public static Dictionary<int, RGBPixel> GetColorPallette(Dictionary<int, List<RGBPixel>> clusters)

{

// for every member of cluster sum all values and get the mean for the sum

Dictionary<int, RGBPixel> colorPallete = new Dictionary<int, RGBPixel>();

foreach (int clusterIndex in clusters.Keys)

{

int sumRed = 0, sumGreen = 0, sumBlue = 0;

int numberOfColorsInCluster = clusters[clusterIndex].Count;

foreach (RGBPixel pixel in clusters[clusterIndex])

{

sumRed += pixel.red;

sumBlue += pixel.blue;

sumGreen += pixel.green;

}

sumRed = (int)Math.Ceiling((double)sumRed / numberOfColorsInCluster);

sumGreen = (int)Math.Ceiling((double)sumGreen / numberOfColorsInCluster);

sumBlue = (int)Math.Ceiling((double)sumBlue / numberOfColorsInCluster);

byte red = Convert.ToByte(sumRed);

byte green = Convert.ToByte(sumGreen);

byte blue = Convert.ToByte(sumBlue);

RGBPixel representitaveColor = new RGBPixel(red, green, blue);

colorPallete.Add(clusterIndex, representitaveColor);

}

return colorPallete;

}

Final order: **Ø(N)**