Image Quantization

(Documentation and Detailed Analysis)

Team: #137

Made By:
Ehab Mohamed Abdelmonem
Gamal Ahmed Abdelhakm
Anas Ahmed Mohamed

1) Graph Construction (Description and Code) $O(N^2)$

V ➔ List of RGBPixel that stores distinct colors

Any color in the image is converted to a shifted version of itself (Red + Green * 1000 + Blue*1000*1000)
The color then is assigned as an index in the following array:
distinct-map[] ➔ Integer Array **O(1)**

As we loop over the image matrix, if it's the first time the color occurs, we mark:
distinct-map[color] ➔ the order of the color in the distinct colors (will be used as a reference) $O(N^2)$

We calculate the distance between distinct colors later while building the MST

Code Snippet:

```
//1 - calculate distinct colors
        height = GetHeight(ImageMatrix);
        width = GetWidth(ImageMatrix);
        V = new List<RGBPixel>();
        distinct_map = new int[256256256];

        // O(N^2)
        for (int i = 0; i < height; i++)
        {
           for (int j = 0; j < width; j++)
           {
              int distinct_color_index = ImageMatrix[i, j].red + ImageMatrix[i, j].green * 1000 +
ImageMatrix[i, j].blue * 1000000;
              if (distinct_map[distinct_color_index] == 0)
              {
                 distinct_map[distinct_color_index] = V.Count+1;
                 V.Add(ImageMatrix[i, j]);
              }
           }
        }
```

2) Calculating Edges + MST Build (Description and Code) **$O(D^2)$**

Using Prim's Algorithm as follows:

Looping over all vertices **O(D)**

Calculating the cost with all other vertices that are not visited before **O(1)**

Getting the minimum cost (distance), store its vertex as adjacent to the initial vertex and mark it as visited **O(D)**

Adding constructed edge to the MST **O(1)**

Code Snippet:

```
//2 - calculate MST ( Prim Algorithm )
        D = V.Count;
        E = new List<KeyValuePair<KeyValuePair<int, int>, double>>();

        var lowest_cost = new double[D];
        var lowest_cost_parent = new int[D];
        for (int i = 0; i < D; i++) lowest_cost[i] = double.MaxValue;

        var is_visited = new bool[D];

        double MST_Sum = 0;
        int current_parent = 0;
        is_visited[0] = true;

        // O(D^2)
        for (int e = 0; e < D - 1; e++) // O(D)
           {
           double minimum_cost = double.MaxValue;
           int minimum_v = 0;
           int current_lowest_cost_parent = 0;
           for (int v = 1; v < D; v++) // O(D)
           {
              if(is_visited[v] == false)
              {
                 double new_cost = Math.Sqrt(((V[current_parent].red - V[v].red) *
(V[current_parent].red - V[v].red)) + ((V[current_parent].green - V[v].green) * (V[current_parent].green
- V[v].green)) + ((V[current_parent].blue - V[v].blue) * (V[current_parent].blue - V[v].blue)));

                 // update lowest cost and parent
                 if (lowest_cost[v] > new_cost)
                 {
                    lowest_cost[v] = new_cost;
                    lowest_cost_parent[v] = current_parent;
                 }

                 // calc gloable lowest cost, parent and child
                 if (lowest_cost[v] < minimum_cost)
                 {
                    minimum_cost = lowest_cost[v];
                    minimum_v = v;
                    current_lowest_cost_parent = lowest_cost_parent[v];
                 }
              }
           }
           E.Add(new KeyValuePair<KeyValuePair<int, int>, double>(new KeyValuePair<int,
int>(current_lowest_cost_parent, minimum_v), minimum_cost));
           MST_Sum += minimum_cost;
           current_parent = minimum_v;
           is_visited[minimum_v]=true;
        }
```

3) Clustering and Palette Generation (Description and Code) **O(K(D + E)) = O(KD)**

Sorting the MST ascendingly **O(D log(D))**

Building the adjacency matrix to include **ONLY** edges needed to cluster “Minimum Distance Choice” (D - K) **O(D - K)**

Applying the BFS Algorithm to each vertex of the adjacency matrix to find each cluster **O(E + D)**

Adding the representative color of each cluster to the color palette as soon as a cluster is generated

Code Snipped:

```
//Make adj matrix
        E.Sort((x, y) => (x.Value.CompareTo(y.Value)));
        var adj = new List<int>[D];
        for (int i = 0; i < D; i++) { adj[i] = new List<int>(); }
        for (int i = 0; i < D - K; i++)
        {
            int c1 = E[i].Key.Key;
            int c2 = E[i].Key.Value;
            adj[c1].Add(c2);
            adj[c2].Add(c1);
        }

        //Calc Clusters
        int class_number = 0;
        var Clusters_Map = new int[D];
        var Clusters_Colors = new List<RGBPixel>();
        var is_Visited = new bool[D];

        // O(K*(D+E)) ---> O(K*D)
        for (int v = 0; v < D; v++)
        {
            if (!is_Visited[v]) // K
            {
                int current_v_temp = v;
                int R = 0, G = 0, B = 0, N = 0;
                Queue<int> neighbors = new Queue<int>();
                neighbors.Enqueue(v);

                //BFS Algo --> D+E
                while (neighbors.Count != 0)
                {
                    v = neighbors.Dequeue();
                    is_Visited[v] = true;
                    R += V[v].red;
                    G += V[v].green;
                    B += V[v].blue;
                    N++;
                    Clusters_Map[v] = class_number;

                    for (int e = 0; e < adj[v].Count; e++)
                    {
                        if (!is_Visited[adj[v][e]]) neighbors.Enqueue(adj[v][e]);
                    }
                }

                RGBPixel new_color = new RGBPixel();
                new_color.red = (byte)(R / N);
                new_color.green = (byte)(G / N);
                new_color.blue = (byte)(B / N);
                Clusters_Colors.Add(new_color);
                class_number++;
                v = current_v_temp;
            }
        }
```