**Team 54**

**Image quantization project**

**Milestone one documentation**

* Hesham Ahmed Mohamed El-Bahey 19
* Hesham Fathy Abdelrazeq Mohamed 19
* Ahmed Salah El-deen Ahmed Mahrous 2
* Mohamed Ahmed Ismael Ahmed 13
* Mohamed Samir Mohamed Kamel 14

**Used structures:-**

1. **Edge: -** contains parent node, destination node and the weight of edge between them.

**Used classes:-**

1. **VertexParent: -** contains current node(V) and parent node(P).

**Used functions:-**

public static List<int> GetDistinctPixels(PixelRGB[,] ImageMatrix);

**Explanation:-**

Extracting distinct colors from ImageMatrix array by looping on it and mapping an encrypted integer value of each PixelRGB to a hashset data structure to prevent repetition of color values.

Encrypting the value of color is made by the following eqn:-

**Encrypted value = (Red value shifted left by 16 + Green value shifted left by 8 + Blue value) .**

Then turning the hashset to a list for the ease of use later on.

Then returning the list of distinct colors.

**Code:-**

public static List<int> GetDistinctPixels(PixelRGB[,] ImageMatrix)

{

int width = ImageMatrix.GetLength(1);

int height = ImageMatrix.GetLength(0);

int r, g, b;

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

for (int y = 0; y < height; y++)

{

for (int x = 0; x < width; x++)

{

r = ImageMatrix[y, x].red;

g = ImageMatrix[y, x].green;

b = ImageMatrix[y, x].blue;

S.Add((r << 16) + (g << 8) + b);

}

}

List<int> L = S.ToList();

return L;

}

**Analysis of the code:-**

Looping :- O(N^2) 🡪 N is the width or height of ImageMatrix.

Hashing :- O(1) 🡪 hashset .Add() .

Turning to list :- O(D) 🡪 D is the number of distinct colors.

Total :- O(N^2) + O(1) + O(D) = O(N^2) .

===============================================

private static int CalcWeight(VertexParent V1,VertexParent V2);

**explanation:-**

Calculates the weight of the edge between two vertices by decrypting the integer values of their color value back to the RGB value and implementing the eqn :-

**Weight = (R2-R1)^2 + (G2-G1)^2 + (B2+B1)^2 .**

**Code:-**

private static int CalcWeight(VertexParent V1, VertexParent V2)

{

byte red1, red2;

byte green1, green2;

byte blue1, blue2;

red1 = (byte)(V1.V >> 16);

red2 = (byte)(V2.V >> 16);

green1 = (byte)(V1.V >> 8);

green2 = (byte)(V2.V >> 8);

blue1 = (byte)(V1.V);

blue2 = (byte)(V2.V);

return (red2 - red1) \* (red2 - red1) + (green2 - green1) \* (green2

-green1) + (blue2 - blue1) \* (blue2 - blue1);

}

**Analysis of the code:-**

**O(1).**

**===============================**

public static List<Edge> PrimMST(List<int> D)**;**

**explanation:-**

Building the MST chain using the list of the distinct colors(D) .

Using Prim's algorithm and FastPriorityQueue (linked library).

By adding the minimum priority to the MST each time.

**Code:-**

public static List<Edge> PrimMST(List<int> D)

{

List<Edge> MSTList = new List<Edge>();

// final list contains the MST

FastPriorityQueue < VertexParent > FP= new FastPriorityQueue<VertexParent>(D.Count);

// Priority queue sorts the priority of edges' weights each time .

VertexParent [] VP = new VertexParent[D.Count];

// array holding each node and it's parent node.

VP[0] = new VertexParent(D[0], null);

// initializing the first node in the MST.

FP.Enqueue(VP[0], 0);

// inserting the first node into the priority queue.

int w;

for (int i = 1; i < D.Count; i++)

{

// intializing all the weights with OO value.

VP[i]=new VertexParent(D[i],null);

FP.Enqueue(VP[i],int.MaxValue);

}

while (FP.Count != 0)

{

VertexParent Top = FP.Dequeue(); // get the minimum priority

if (Top.P != null) // if it is not the starting node.

{

Edge E;

E.V1 = Top.V;

E.V2 = (int)(Top.P);

E.Weight = (int)(Top.Priority);

MSTList.Add(E); // add the minimum weight to the MST.

}

foreach (var unit in FP) // modify the priority each time .

{

w = CalcWeight(unit, Top);

// calculates the weight between the current node and the top node.

if (w < unit.Priority)

{

unit.P = Top.V;

FP.UpdatePriority(unit, w);

}

}

}

return MSTList;

}

**Analysis of the code: -**

Initializing priorities of each vertex: - O(D) 🡪 D the number of distinct colors.

Prim's Algorithm: - O(DElog(D)) 🡪 for each priority queue vertex O(D)\*modifying the priority O(log(D)) \* for each edge O(E) .

Total: - O(D) + O(DElog(D)) = O(DElog(D)).

Thank you…