Description: Logo

Description automatically generated

A picture containing text

Description automatically generated

Algorithems Document



Team ID (T079) – Image Quantization

TA. Mohamed Magdi

Team Members: -

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | ID | | Department | Section |
| Osama Anter Mohamed Afify | | 20191700091 | SC | 1 |
| Ahmed Mohamed Ebrahim Mohamed | | 20191700059 | SC | 1 |
| Tarek Ashraf Mahmoud Hussein | | 20191700322 | SC | 3 |

**Classes we use :-**

1. ChildParent: we created this class and it inheritance from

FastPriorityQueueNode and use it as data type in fast queue data type

And it contains (child) and (parent) node.

1. FastPriorityQueue : we created "copied" this class to store struct Edge and sort elements by weight in the queue and to reduce the complexity.

**Structs we use :-**

1 // Holds the pixel color in 3 byte values: red, green and blue

public struct RGBPixel

{

public byte red, green, blue;

}

2 // Holds the pixel color in 3 color in float: red, green and blue

public struct RGBPixelD

{

public double red, green, blue;

}

3 // contains parent node, destination node and the weight of edge between them

public struct Edge

{

//return MST

public float weight;

public int v1, v2;

}

4 // to store graph ,where the p = parent , int 🡪 distnation , float 🡪 weight

public struct test

{

public int p; //parent

public List<KeyValuePair<int, float>> info; // <Child , cost>

}

1. **Graph construction :**

**Code:**

1. public static List<test> ConstructGraph(List<int> distinct\_values)
2. {
3. float result = 0; //O(1)
   * 1. int numberOfDistinct = distinct\_values.Count; //O(1)
4. List<test> test\_output = new List<test>(numberOfDistinct); //O(1)
6. for (int i = 0; i < numberOfDistinct; i++)// //O(n^2)
7. {
8. byte r1 = (byte)(distinct\_values[i] >> 16); //O(1)
9. byte g1 = (byte)(distinct\_values[i] >> 8); //O(1)
10. byte b1 = (byte)(distinct\_values[i]); //O(1)
11. test res = new test();//O(1)
12. res.p = distinct\_values[i]; //O(1)
13. List<KeyValuePair<int, float>> res\_list = new List<KeyValuePair<int, float>>(); //O(1)
14. List<KeyValuePair<int, float>> temp\_list = new List<KeyValuePair<int, float>>(); //O(1)
15. List < Dictionary<int, float> > edge\_list = new List<Dictionary<int, float>>(); //O(1)
16. for (int j = i + 1; j < numberOfDistinct; j++) //n
17. {
18. if (i == j) //O(1)
19. {
20. continue; //O(1)
21. }
22. byte r2 = (byte)(distinct\_values[j] >> 16); //O(1)
23. byte g2 = (byte)(distinct\_values[j] >> 8); //O(1)
24. byte b2 = (byte)(distinct\_values[j]); //O(1)
25. result = (float)Math.Sqrt(((r1 - r2) \* (r1 - r2)) + ((g1 - g2) \* (g1 - g2)) + ((b1 - b2) \* (b1 - b2))); //O(1)
27. KeyValuePair<int, float> out\_dict = new KeyValuePair<int, float>(distinct\_values[j], result); //O(1)
28. //out\_dict.Add(distinct\_values[j], result);
29. res\_list.Add(out\_dict); //O(1)
30. //check if that the minimum edge
31. }
32. res.info = res\_list; //O(1)
33. test\_output.Add(res); //O(1)
34. }
35. return test\_output; //O(1)
36. }

**Description :**

This function takes the number of colors of the original image “distinct\_values ” without repetition and we take it in a list, then we follow these times, color by color. Then we connect these pixels to each other in the form of a graph and calculate of the “cost” between them each time. Then we come back at the end in List of test Struct as graph.

**Code Analysis :**

**Total order : O((distinct\_values.Count) ^2) = O(N^2).**

**2-Minimum spanning tree code:**

//extrct minumum spaning tree

public static List<Edge> extrctMSTFromGraphByPrim(List<test> output, List<int> distinct\_values)

{

int sizeOfGraph = distinct\_values.Count; //O(1)

//O(1)

FastPriorityQueue<ChildParent> toSortedGraph = new FastPriorityQueue<ChildParent>(sizeOfGraph);

List<Edge> MSTFromGraph = new List<Edge>(); //O(1)

for (int i = 0; i < sizeOfGraph; i++) // O(n)

{

if (i == 0) //O(1)

{

toSortedGraph.Enqueue(new ChildParent(-1, distinct\_values[0]), 0);//O(log(n))

}

else

{

toSortedGraph.Enqueue(new ChildParent(-1, distinct\_values[i]), float.MaxValue);//O(log(n))

}

}

ChildParent minCP; //O(1)

while (true) //while only O(1) becouse it fixed count but why body //O(nlog(n))

{

minCP = toSortedGraph.Dequeue();//O(1)

if (minCP.parent >= 0)

{

Edge edgetmp;//O(1)

edgetmp.v1 = minCP.parent;//O(1)

edgetmp.v2 = minCP.child;//O(1)

edgetmp.weight = minCP.Priority;//O(1)

MSTFromGraph.Add(edgetmp);//O(1)

}

foreach (var item in toSortedGraph) //O(N)

{

// to calculate distance btween tow pixels

byte redV1, redV2, greenV1, greenV2,blueV1, blueV2;//O(1)

redV1 = (byte)(minCP.child >> 16);//O(1)

redV2 = (byte)(item.child >> 16);//O(1)

greenV1 = (byte)(minCP.child >> 8);//O(1)

greenV2 = (byte)(item.child >> 8);//O(1)

blueV1 = (byte)(minCP.child);//O(1)

blueV2 = (byte)(item.child);//O(1)

float Distance = (redV2 - redV1) \* (redV2 - redV1) + (greenV2 - greenV1) \* (greenV2 - greenV1) + (blueV2 - blueV1) \* (blueV2 - blueV1);//O(1)

float cal\_w = (float)Math.Sqrt(Distance);//O(1)

// end

if (cal\_w < item.Priority)

{

item.parent = minCP.child;//O(1)

toSortedGraph.UpdatePriority(item, cal\_w);//O(log(n))

}

}

if (toSortedGraph.Count == 0)//O(1)

{

break;//O(1)

}

else

{

continue;//O(1)

}

}

return MSTFromGraph;//O(1)

}

**Description :**

This function takes the number of colors of the original image “distinct\_values” without repetition and graph that construct from "ConstructGraph" , then we use Prim's algorithm and FastPriorityQueue to building the MST chain using the list of the distinct colors and graph by adding the minimum priority to the MST each time.

**Code Analysis :**

**Total order : O(N\*log(N)) + O(N\*log(N)) = O(N\*log(N)).**

3- Palette generation code :

1. //color palette
2. Dictionary<int, int> palette\_dict = new Dictionary<int, int>();
3. //List<int> temp = new List<int>();
4. Dictionary<int, int> PalleteRes = new Dictionary<int, int>();
5. //acess list of clusters
6. foreach (var v in clusters)
7. {
8. int red = 0, green = 0, blue = 0;
9. int average = 0;
10. for (int i = 0; i < v.Count; i++)
11. {
12. palette\_dict.Add(v.ToList()[i], (byte)(v.Average()));
13. //temp.Add((byte)(v.Average()));
14. }
15. }
16. foreach (var set in clusters)
17. {
18. int red = 0, green = 0,blue = 0;
19. int value = 0;
20. foreach (var unit in set)
21. {
22. green = (green + (byte)(unit >> 8));
23. red = (red + (byte)(unit >> 16));
24. blue = (blue + (byte)(unit));
25. }
26. red = (red / set.Count);
27. green = (green / set.Count);
28. blue = (blue / set.Count);
29. value = (red << 16) + (green << 8) + (blue);
30. foreach (var unit in set)
31. {
32. PalleteRes.Add(unit, value);
33. }
34. }
35. return PalleteRes;

**Code Analysis :**

**Total order : O(D)**

**Thank YOU Dr**