**Image Quantization**

**T 099**

ابانوب جما ل فكري فوزي SC 20191700001

شادي رافت فايق توفيق CSYS 20191700311

مايكل مجدي فكري شحاته IS 20191700487

Graph Description And Code.

//////////////////////////////////////

//Graph construction

//////////////////////////////////////

int Height = GetHeight(ImageMatrix);

int Width = GetWidth(ImageMatrix);

// Checking if the color is visted or not by A new Array (All vis = 0)

int[,,] visted = new int[260, 260, 260];

//Map passing by every colour to select k-cluster

RGBPixel[,,] map = new RGBPixel[260, 260, 260];

// Array selecting Distinct colours

RGBPixel[] distinct = new RGBPixel[6000000];

int ptr = 0;

//complexty = O(N^2) when N is the length of the photo

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

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

{

RGBPixel r = ImageMatrix[i, j];

if (visted[r.red, r.green, r.blue] == 0)

{

distinct[ptr++] = ImageMatrix[i, j];

visted[r.red, r.green, r.blue] = 1;

}

}

Graph description  
-Taking initial width and height from the image matrix

-initialize 3D array for the three color type red green and blue

-Making A map for the three colors ,

-initialize Empty array to save in it the distinct colors of image in it with max value 6000000

-make a pointer to points on last value on the nested loop  
Nested loop (first less than Height of image),(second less than width of image)  
initialize the first position in new pixel array with Index of first and second loop(i,j)  
check if the 3D visited array isn’t visited   
then placement of pointer of next index by the (I,j) position   
then making the 3D array of visited to be visited   
-Analysis

complexity = o(N^2)

Minimum spanning tree code.

///////////////////////////////////////////////

//MST Code impilimentation

//////////////////////////////////////////////

ArrayList[] adj = new ArrayList[ptr];

// to store the colours in the same cluster

ArrayList[] cluster = new ArrayList[ptr];

// O(N) when N is the number of distincit colours

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

{

adj[i] = new ArrayList();

cluster[i] = new ArrayList();

}

// visited array to check the node aded for the mst or not

int[] vis = new int[ptr];

int n = ptr;

edje[] mst = new edje[n - 1];

int indx = 0;

vis[0] = 1;

double sum = 0;

edje[] mn = new edje[n];

edje temp;

temp.w = 1e9;

temp.v = 0;

temp.u = 0;

// first add the first colour to the mst

// O(N) when N is the number of distincit colours

for (int i = 1; i < n; i++)

{

mn[i].w = calc(distinct[i], distinct[0]);

mn[i].u = i;

mn[i].v = 0;

}

vis[0] = 1;

mn[0].w = 1e9;

// O(N^2) when N the number of distincit colours

while (true)

{

// every time add one node with the lowest cost for mst

bool f = false;

temp.w = 1e9;

for (int i = 1; i < n; i++)

{

if (temp.w > mn[i].w)

{

temp = mn[i];

f = true;

}

}

// if i didnt add any node i sure that the mst completed and break

if (!f)

break;

// update the minmum edje for every node

mst[indx++] = temp;

sum += temp.w;

vis[temp.u] = 1;

mn[temp.u].w = 1e9;

for (int i = 1; i < n; i++)

{

if (vis[i] == 1)

continue;

edje temp2;

temp2.w = calc(distinct[temp.u], distinct[i]);

temp2.u = i;

temp2.v = temp.u;

if (mn[i].w > temp2.w)

mn[i] = temp2;

}

}

-Analysis

complexity = O(N^2)

Palette generation code.

/////////////////////////////////////////

///Palette Generation (k-Clusters)

//////////////////////////////////////////

// O(N\* log(n)) N the number of distincit colours

///Sort by Launda expression

Array.Sort(mst, (x, y) => y.w.CompareTo(x.w));

// Saving K-clusters

int kc = (int)sigma;

// Colours of index To be used

RGBPixel[] K\_Colours = new RGBPixel[kc];

indx = 0;

// Remove the biggest edges of the colour k to make the MST k clusters

for (int i = kc - 1; i < n - 1; i++)

{

int u = mst[i].u, v = mst[i].v;

//Adding each colour From the spaning tree

adj[u].Add(v);

adj[v].Add(u);

}

//Make a new non-visted array

vis = new int[ptr];

// O(N) the number of distincit colours

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

{

//Check if visted

if (vis[i] == 1)

continue;

//Enqueue Each visted vertex of the tree

Queue<int> qe = new Queue<int>();

qe.Enqueue(i);

//////////

while (qe.Count > 0)

{

int node = qe.Dequeue();

cluster[indx].Add(node);

vis[node] = 1;

foreach (int j in adj[node])

{

if (vis[j] == 1)

continue;

vis[j] = 1;

qe.Enqueue(j);

}

}

indx++;

////////////////

}

indx = 0;

// O(N^2) N the number of distincit colours

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

{

// loop for every clusters and select the colour with max distance is minmum

double w = 1e9;

int center = 0;

//checking on each cluster

foreach (int j in cluster[i])

{

double cur = 0;

//Check to get max distance between each cluster

foreach (int k in cluster[i])

{

cur = Math.Max(cur, calc(distinct[j], distinct[k]));

}

//Exchange Each MAX Distance Between Each Cluster

if (cur < w)

{

w = cur;

center = j;

}

}

K\_Colours[indx++] = distinct[center];

}

// map every colour to new one

// O(N\*D)

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

{

// chose The Best colour (Minimum Available Colour)

double w = 1e9;

RGBPixel col;

col.red = 0;

col.green = 0;

col.blue = 0;

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

{

if (calc(distinct[i], K\_Colours[j]) < w)

{

w = calc(distinct[i], K\_Colours[j]);

col = K\_Colours[j];

}

}

//Adding each distinct colour at the map

map[distinct[i].red, distinct[i].green, distinct[i].blue] = col;

}

// print the final Cluster Colour

// O(N^2)

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

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

{

int r = ImageMatrix[i, j].red;

int g = ImageMatrix[i, j].green;

int b = ImageMatrix[i, j].blue;

ImageMatrix[i, j] = map[r, g, b];

}

-Analysis

complexity = O(N^2)

**Time of All Test:**

|  |  |
| --- | --- |
| Sample Test Case | Time In (second) |
| Sample.Case1 | 0.008 |
| Sample.Case2 | 0.004 |
| Sample.Case3 | 0.136 |
| Sample.Case4 | 0.018 |
| Sample.Case5 | 0.01 |

|  |  |
| --- | --- |
| COMPLETE Test Case | Time In (second) |
| Small.Case1 | 1.927 |
| Small.Case2 | 2.03 |
| Medium.Case1 | 19.577 |
| Medium.Case2 | 7.068 |
| Large.Case1 | 52.395 |
| Large.Case2 | 70.416 |