## Image Quantization project

# Problem Definition

## What is Image Quantization?

The idea of *color quantization* is to reduce the number of colors in a full resolution digital color image (24 bits per pixel) to a smaller set of representative colors called ***color palette*.** Reduction should be performed so that the quantized image differs as little as possible from the original image. Algorithmic optimization task is to find such a color palette that the overall distortion is minimized.

What is our project do?

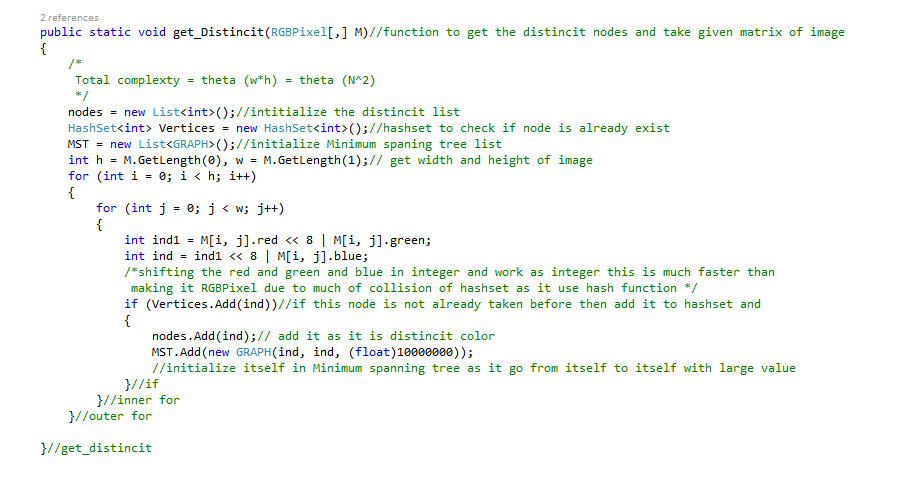
The idea of our project to quantize an image to be a little different.

The main steps:

1. Get the distinct colors of the image
2. Build a graph (MST) to connect each node with its child.
3. Clustering every similar colors in a group called cluster.
4. Get an average color of each cluster.
5. Distributing each specific color at its own place.
6. Get the distinct colors of the image
7. Description:

We need to compare colours of each cell of the given matrix so we need data type that can hold the three colours at the same time so we need to shift bytes so that we could have integer hold the entire three colours. And we already taking the three colours of each cell with the given matrix.

1. Complexity : O(d^2)
2. code



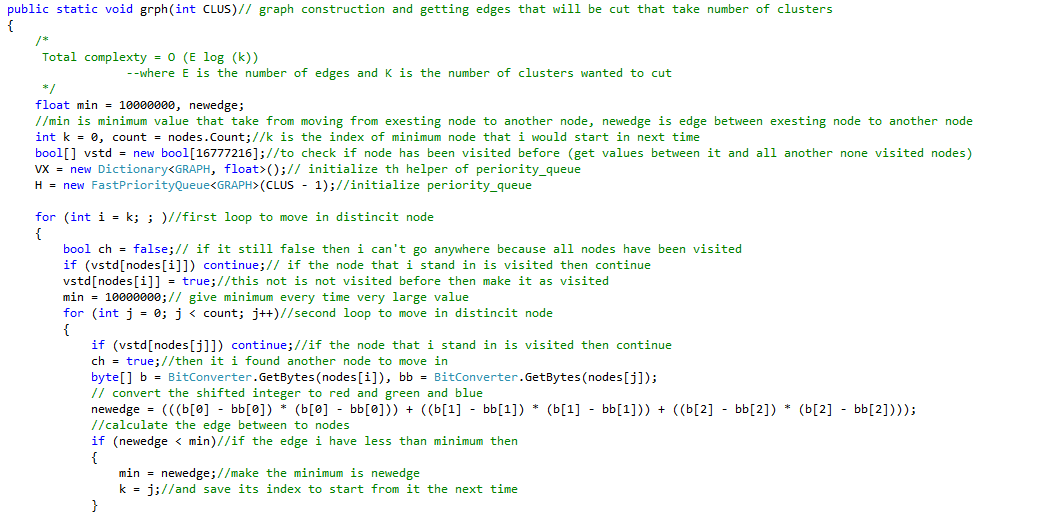
1. Build a graph (MST) to connect each node with its child
2. Description:

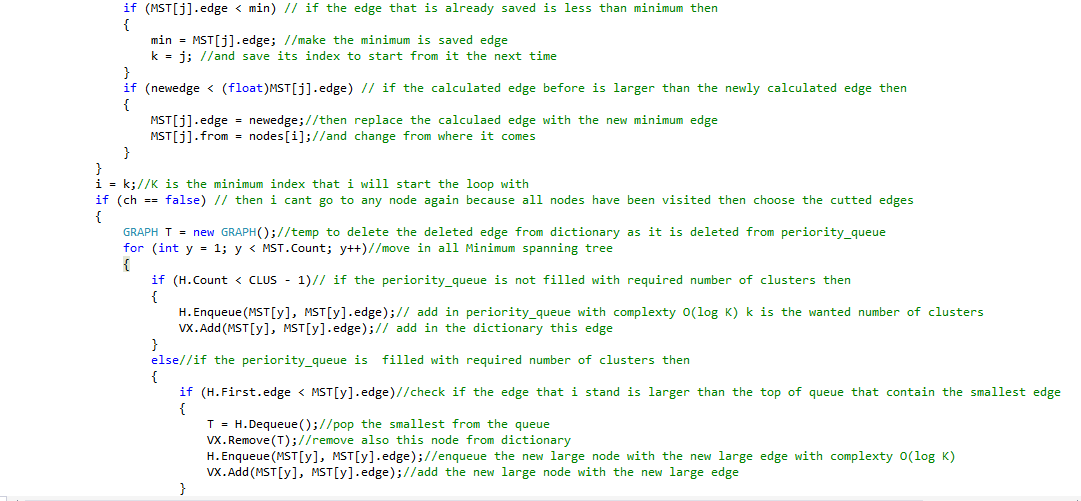
The idea to hold every node and get the shortest path to the next node to create a minimum spanning tree holds the all tree.

1. Complexity:

O(E log K) E is the number of edges and K is number of clusters

1. Code:

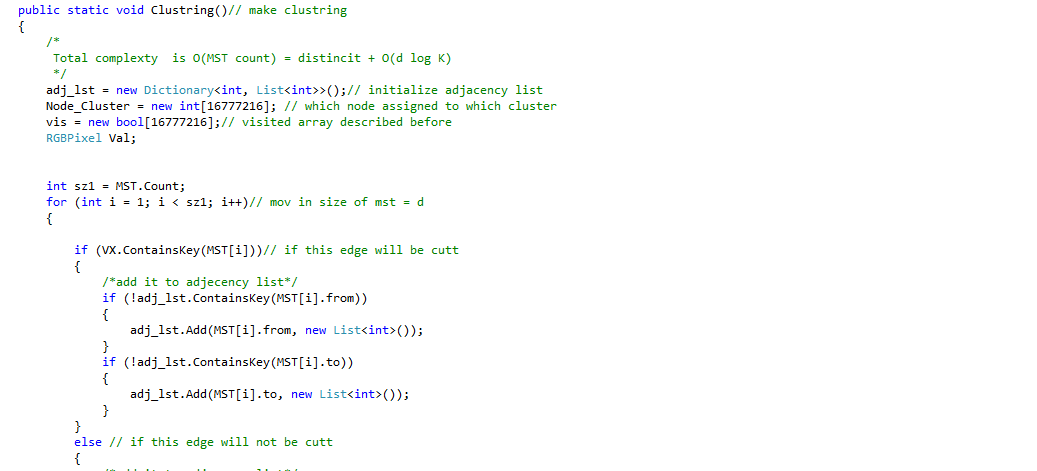


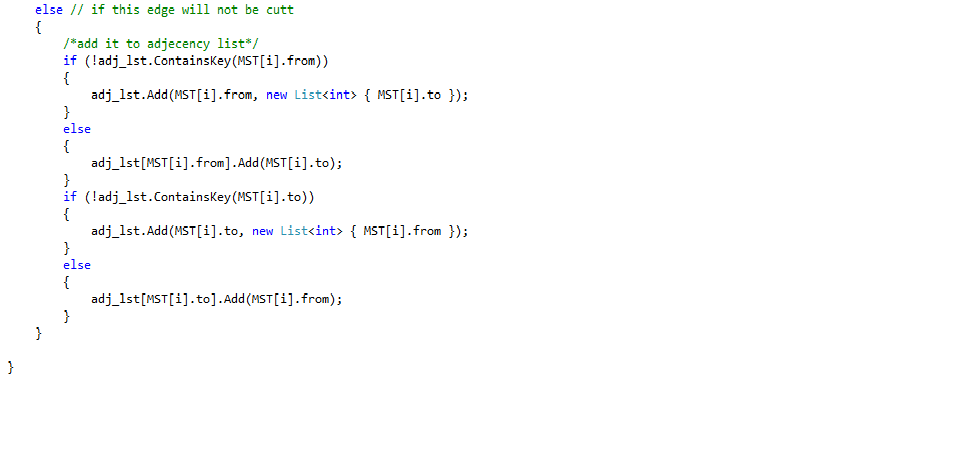


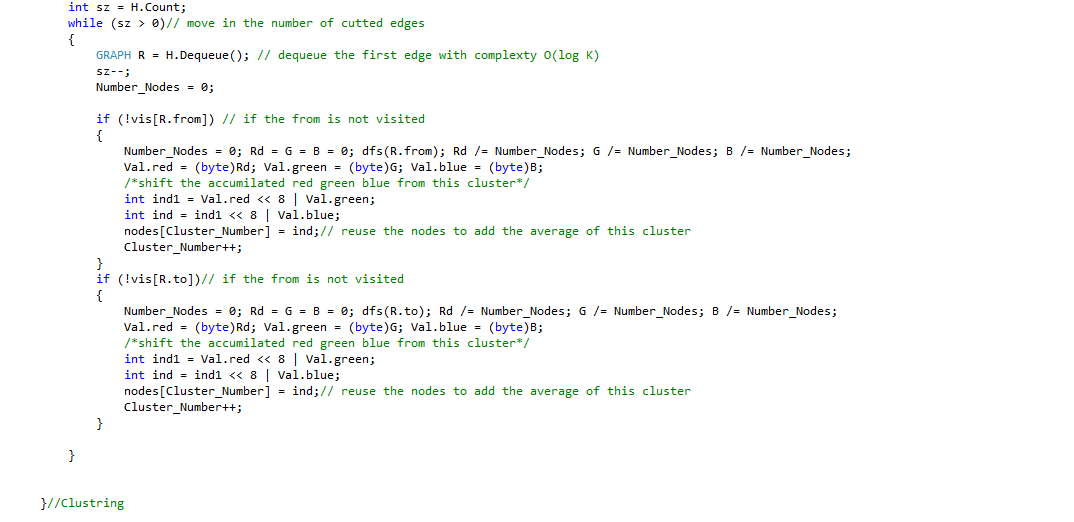
1. Clustering every similar colors in a group called cluster.
2. Description:

The idea is to hold the maximum edges in the (MST) and cut on them to number of clusters, each cluster have its own nodes.­­

1. Complexity: O(MST count) = O(d) + O(d log K)
2. Code:







1. Get an average color of each cluster (DFS).
2. Description:

Assumed that graph (cluster) is connected. Depth-first search visits every vertex in the graph and checks every edge its edge and compute the average color ,the color which have the most common color in the cluster.

1. Complexity:

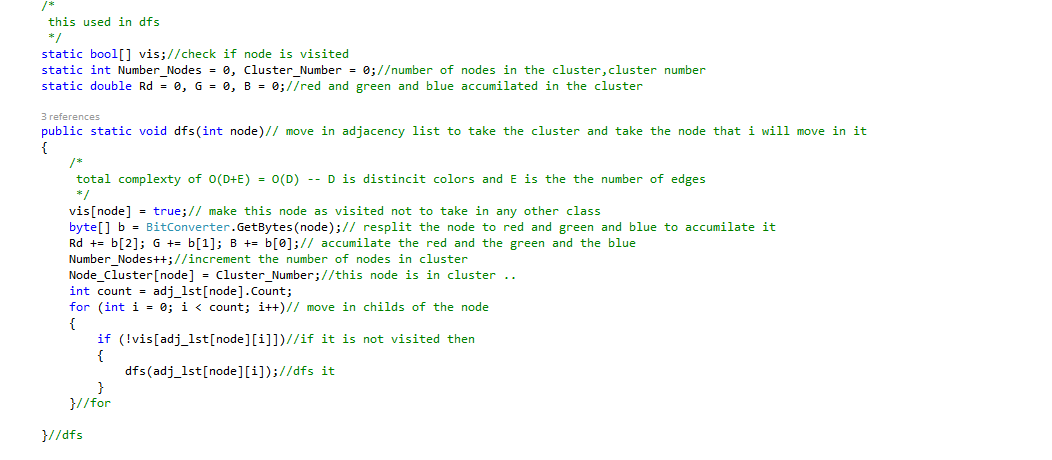
The loop will take O(V) where V is the number of vertices

Each node is visited once since we make a visited array not to step again in the same node

Visit is executed |Adj[*v*]| times. The total cost of executing DFS-Visit is ∑*v*∈*V*|Adj[*v*]| = Θ(*E*)

Total running time of DFS isΘ(*V+E*)*.*

1. Code:



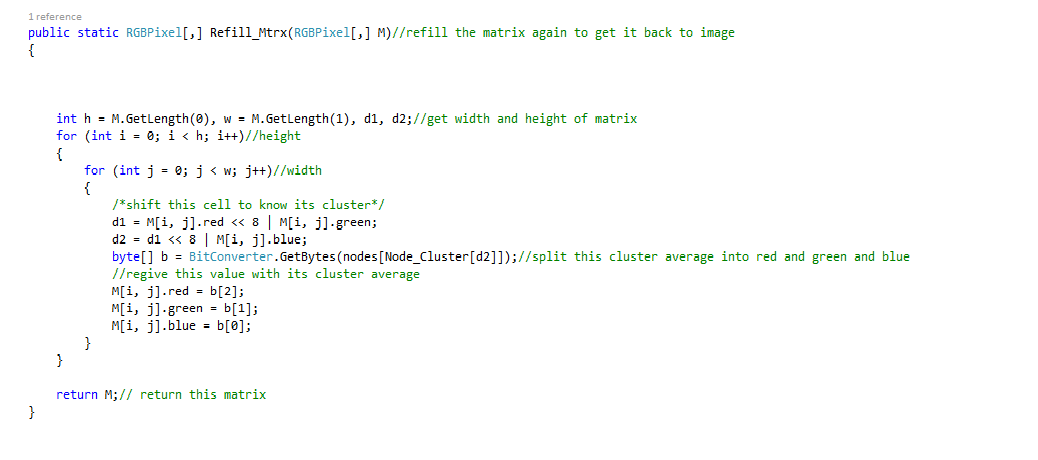
1. Distributing each specific color at its own place.
2. Description:

After finishing clustering we need to get back every color(the new color which computed in dfs ) in its position in the image .

1. Complexity:

O(N^2) N is the width \* height of given matrix

1. Code:



Fast priority queue :

It contains :-

\_numNodes as the number of nodes

And

T[] \_nodes as the values of queue

public int Count

{

get

{

return \_numNodes;

}

}

It is the number of nodes contained in the priority queue complexty O(1)

public void Enqueue(T node, float priority)

it is the function that add values to the queue with complexty o(log n) n is the number of elements in the queue.

After adding the element it check its heap property by calling cascade up that if the added element is larger than its root shifting or cascading up appears.

private void CascadeUp(T node)

{

//aka Heapify-up

int parent = node.QueueIndex / 2;

while(parent >= 1)

{

T parentNode = \_nodes[parent];

if(HasHigherPriority(parentNode, node))

break;

//Node has lower priority value, so move it up the heap

Swap(node, parentNode);

parent = node.QueueIndex / 2;

}

}

Complexty O(log N) ) n is the number of elements in the queue.

Cascade function use the function

private bool HasHigherPriority(T higher, T lower)

{

return (higher.Priority <= lower.Priority);

}

That compare between periorities of 2 elements with complexty O(1).

public T Dequeue()

It is function that remove element from the queue with complexty O(log N) n is the number of elements in the queue.

It uses function Remove

public void Remove(T node) that have complexty of O(1)+ complexty of function OnNodeUpdated.

private void OnNodeUpdated(T node)

Function that rebuild the queue correctly by Cascading Up or Cascading down .

private void CascadeDown(T node)

Function That change the rebuild the queue correctly to keep the heap property in the queue with complexty O(log K)

public T First

{

get

{

#if DEBUG

if(\_numNodes <= 0)

{

throw new InvalidOperationException("Cannot call .First on an empty queue");

}

#endif

return \_nodes[1];

}

}

Function that get the top element of the queue with complexty O(1).