Data Structures (Spring-2023)

Deadline: Sunday, 30th April, 2023

Assignment# 04

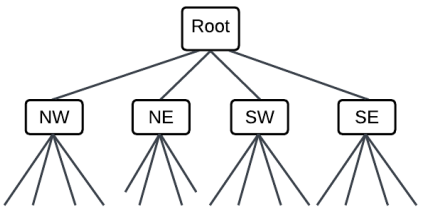
Instructions:

- All the submissions will be done on Google classroom.
- You must submit 3 cpp files in Zip Folder named (21I-XXXX.zip). Naming convention has to be followed strictly. 30% marks will be deducted for not following submission guidelines.
- The student is solely responsible to check the final zip files for issues like corrupt files, viruses in the file, mistakenly exe sent.
- Be prepared for viva or anything else after the submission of assignment.
- Zero marks will be awarded to the students involved in plagiarism.
- Late submission of your solution is not allowed. Submission within 30 minutes after deadline will be accepted with 50% deduction. After that no submission will be accepted.
- Do not use Vectors in this Assignment. Usage of vector will result in straight zero. Also you are not allowed to use any built-in functions provided by image processing libraries except image reading functionality. As images are 2-D arrays, you will process them yourself without any ready-made functions.
- Link to install Opencv on Windows: https://www.youtube.com/watch?v=unSce_GPwto
- Link to install Opencv on Mac OS: https://www.youtube.com/watch?v=Ozc3zWJ_NhQ
- Understanding the assignment is also the part of assignment.
- **You need to make separate cpp file for each part.**

The goal of this assignment is to develop a program that can encode and decode an image using the quad tree and 2d linked list data structures.

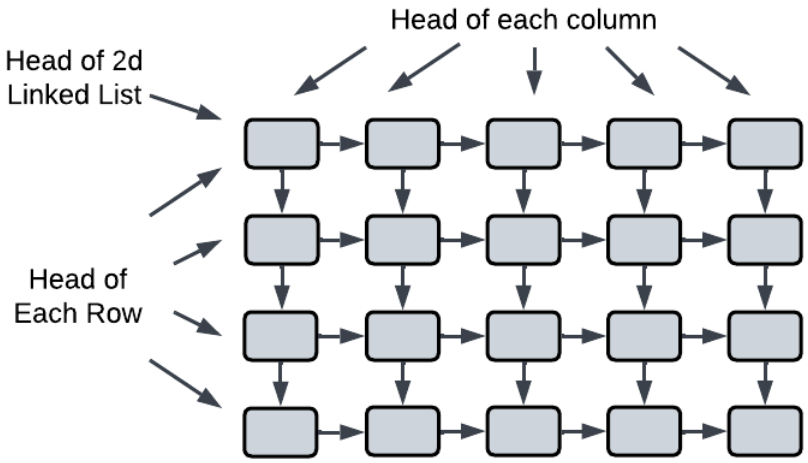
Quad tree:

Quad tree is a tree data structure in which each internal node has exactly four children: north-west (NW), north-east (NE), south-west (SW), and south-east (SE). Each node represents a quadrant in a two-dimensional space. This data structure is used to recursively divide a two-dimensional space into four quadrants until each quadrant contains only one element. In this case, the elements are the pixels in the image. The quad tree is constructed by dividing the image into four quadrants, and recursively applying the same process to each quadrant until each quadrant contains only one pixel.



2d Linked List:

2d Linked List is a data structure that allows the representation of a 2D matrix or grid. It consists of a set of nodes, each of which has two pointers, one pointing to the node on its right and the other pointing to the node below it. The data in each node can be any type of data that needs to be stored. The first node of each row is called the "head" of the row, and the first node of the first row is called the "head" of the entire linked list. Similarly, the first node of each column is called the "head" of the column.



Encoding Process:

Encoding process involves constructing a quad tree for the given image. The image is divided into four quadrants, and each quadrant is recursively divided into four sub-quadrants until each sub-quadrant reach the base condition.

Node in a quad tree represents a square region of an image. Each node has three colors, **White**, **Black** or **Grey** depending on whether the region has all white, all black or mixture of white or black (a 'grey' region) pixels. Each 'grey' region is broken into 4 equal parts. Thus, each grey node has four children representing four equal parts also labeled white, black or grey. The process will be repeated until there is no grey nodes without children. Note that the root node represents the whole image and the leaf nodes are either black or white. Also only grey node has children, not white or black.

Question:

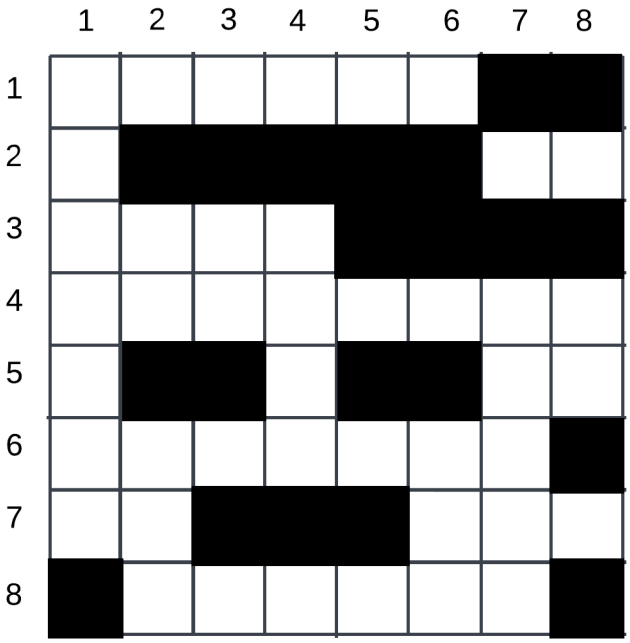
Part 1 –

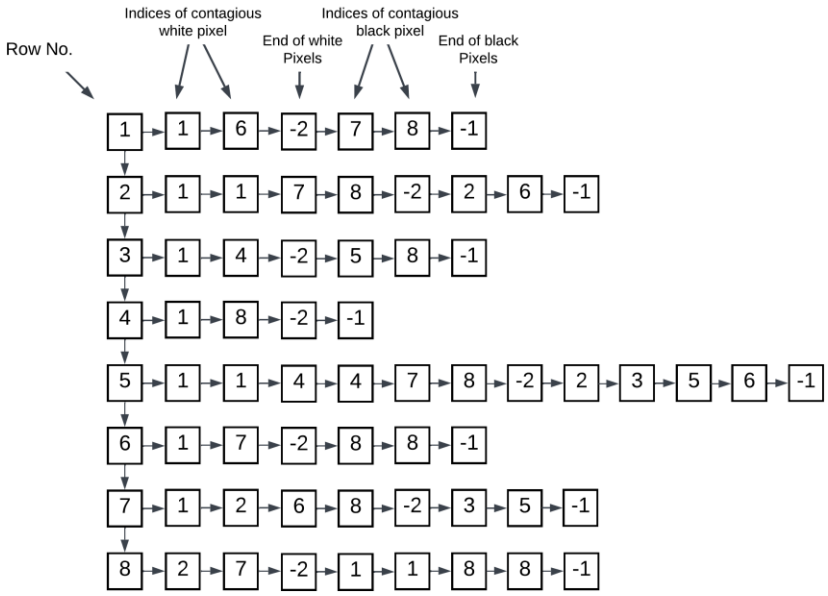
You will write code to compute quad tree of an image. First, implement an abstract data structure of Quad Tree and 2d Linked List. You need to make a member function of 2d linked list named:

void convertTo2dLL (int array, int row, int col)**

This will convert 2d DMA into 2d Linked List. Read an image using opencv library functions into 2d DMA and pass the DMA to **convertTo2dLL** Function.

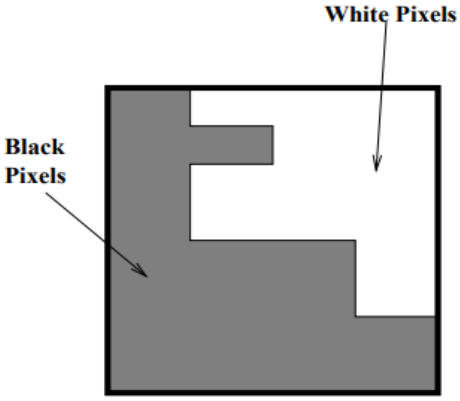
The **convertTo2dLL** Function will convert the image into 2d link list where each row of Link list corresponds to a row of image. In each row, we store the indices of contagious pixel values (White and Black separately). The -2 indicates the end of the white pixels and -1 indicates the end of the black pixels of the row. If there are more than once contagious blocks, we store all the starting and ending column indexes of that row. If row does not have any pixels with required pixel value (white or black), just store their end indicators (-1 or -2).

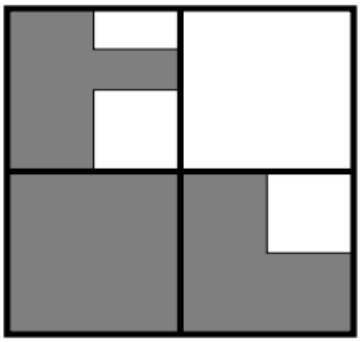


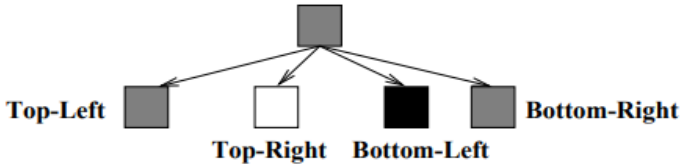


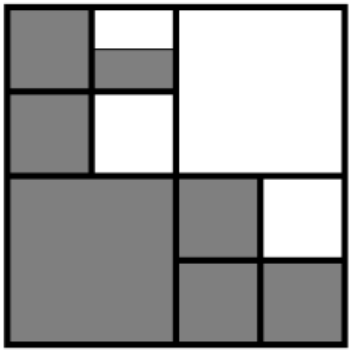
In above pictures, there is an image and how it looks like in 2d linked list. In this case, down pointer will be NULL for the whole row except the first node of each row that contains the row number.

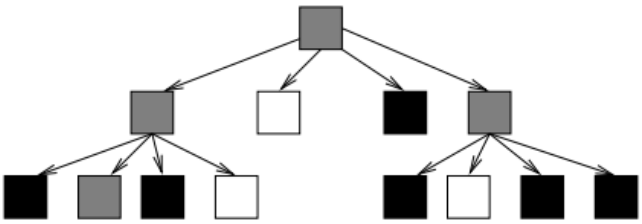
Now, you have a 2d Linked list that contains your image. Pass the 2d Linked list to Quad tree to construct the quad tree for the input image.

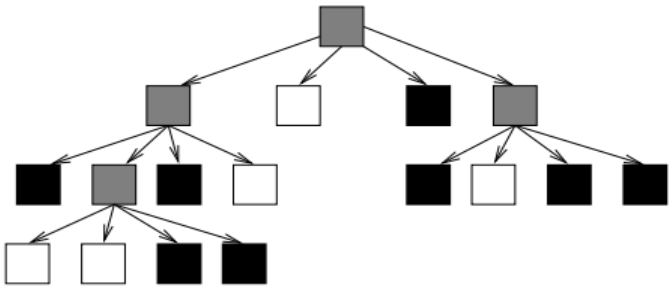












In above picture, stages of building a quad tree is shown. The input image on the left and quad tree is on the right. Divided regions are marked with dark lines on image.

After successfully completing the quad tree, next step is to traverse the tree and write each node in separate file. Make a folder and store all the tree nodes in it. Root node must be of file name **root.** For other node, you can use any naming convention. You must store color, position and address to children with each node. Again it's up to you, how you will store these things in file.

Decoding Process:

Decoding process involves reading each node from the file starting from **root** and reconstructing the quad tree. The image is then constructed by recursively traversing the quad tree and assigning the color of each node to the pixels in the corresponding quadrant.

Part 2 –

You will write code to construct an image from quad tree. First, read each node from the file, and reconstruct the quad tree. Write a **function** to traverse the reconstructed quad tree and regenerate the image (Assign the color of each node to the corresponding pixels in the image (or 2d DMA)). After constructing an image from quad tree, write the image on using **imwrite** function of OpenCV. (Hint: Recursive function is used for traversing a Tree).

Remember that this time image will be in 2d DMA.

Part 3 –

You will write code to find the accuracy of your generated image by comparing the image with original image. First, you need to load the both original and decoded image in 2d DMAs. Then, you need to use the following formulas for finding accuracy.

$$X = \frac{\sum_{i=0}^{n} \left((O(i,j) - D(i,j))^2\right)}{n}$$

Where,

- O(i,j) is the pixel value of original image at (i,j).
- D(i,j) is the pixel value of decoded image at (i,j).
- n = N * M where, N = number of rows & M = number of columns

$$Y = 100 * (1 - (X / Z^2))$$

Where,

- X is from above formula.
- Z is the max pixel value which in your case is 255.

Note:

- You will use 2d linked list in Part 1 only. Part 2 & Part 3 will done using 2d DMA arrays.
- You need to make separate cpp file for each part.
- You can attempt part 3 independently. Just input 2 images and preform the required calculation.

---------- **Happy EID** 😊 ----------