Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

GÖKÇE NUR ERER
171044079

Course Assistant: Özgü Göksü

1 INTRODUCTION

1.1 Problem Definition

The objective is to construct a max priority queue with a binary heap implementation for color pixels in an image. It is required to add the color pixels in to 3 different priority queues each of them requiring either: lexicographical comparison, Euclidean comparison or bitmix comparison.

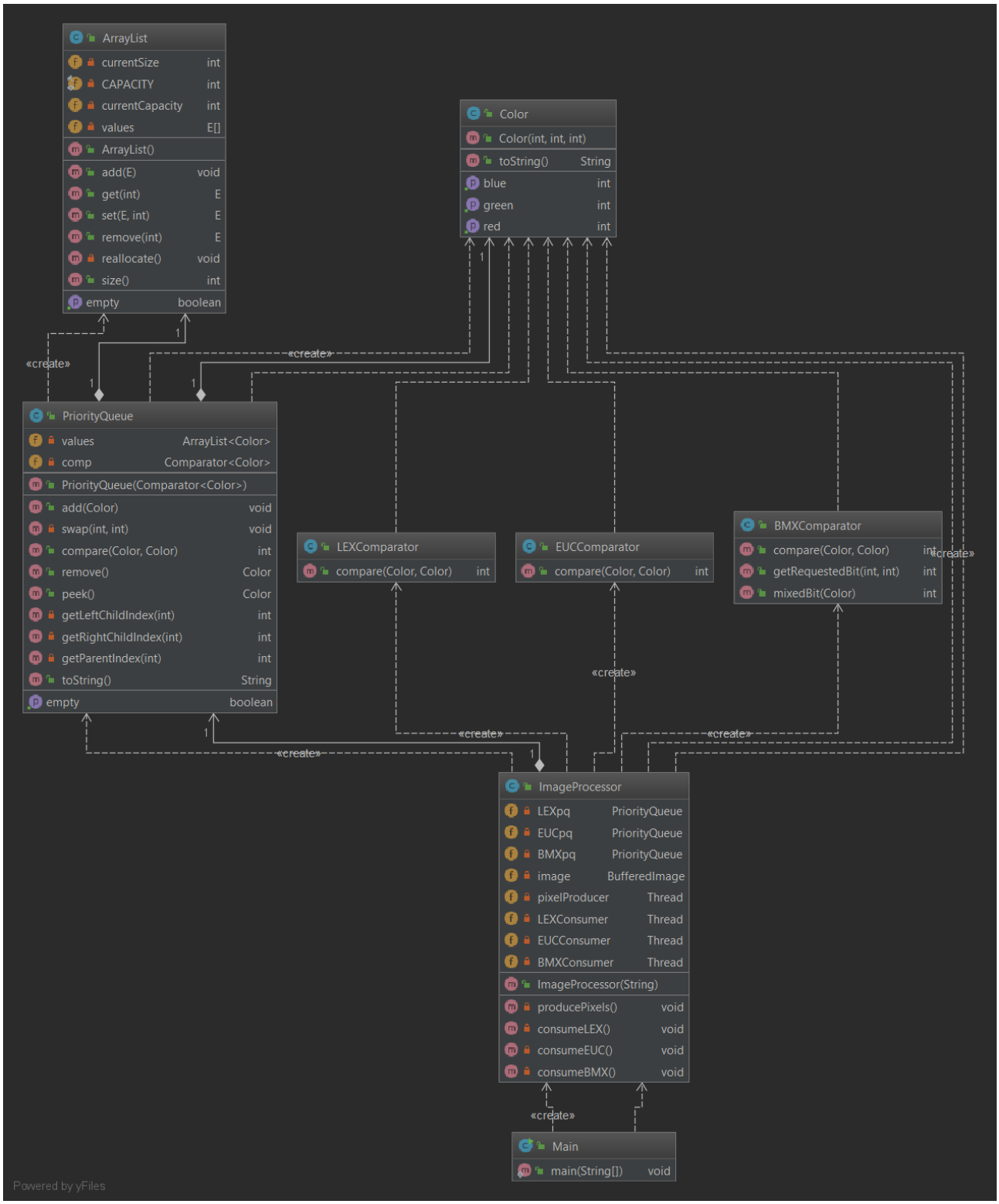
Also it is requested to do the producing the pixels and consuming the pixels from different queues to be run on different threads. Threads should be synchronized.

1.2 System Requirements

This project can run on an average computer which has Java components like (JVM, SDK, JDK...) and has an IntelliJ Idea IDE since it is and IntelliJ project.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams

User is expected to enter the file path that is requested to be processed in this software. Nothing else is requested from the user.

2.3 Problem Solution Approach

While solving this problem seven classes were required:

ArrayList Class: This class implements a basic generic ArrayList. It has the following methods and constructors:

ArrayList ():	creates an arraylist with the capacity of 10 and creates a new inner array with that same capacity.
void add (E entry):	adds the given entry to the end of the arraylist. Complexity for this method is: O(n) if it needs reallocation, O (1) if there is enough capacity to hold a new element.
E get (int index):	gets the entry at the given index Complexity for this method is: O (1) since it is reaching to an element in a normal array.
E set (E entry, int index):	sets the entry at the given index with the given entry Complexity for this method is: O (1) since it is reaching to an element in a normal array and just assigning it to a new value.
E remove (int index):	removes the element at the specified position. Shifts any subsequent elements to the left. Complexity for this method is O (1) if the element is removed from the end of the arraylist. O(n) if the element is removed from the start or anywhere in the middle.

void reallocate (): doubles the capacity, creates a new array with the new capacity. Refills the new array and assigns it to the arraylist's inner array.
Complexity for this method is O(n) since all elements have to be traversed to be added to the new array.

int size (): returns the size of the arraylist.
Complexity for this method is O (1) since it just returns the size value kept in the arraylist.

boolean isEmpty (): returns true if the arraylist is empty, else it returns false.
Complexity for this method is O (1), since it just calls the size () method which has an O (1) complexity.

PriorityQueue Class: This class implements a Priority Queue with a max binary heap implementation inside. It uses an ArrayList and a Comparator to do its operations. It has the following constructors and methods:

PriorityQueue(Comparator<Color> comp) : creates a priority queue with a new empty arraylist and assigns the given comparator to its comparator.

void add (Color c): adds the given color to the priority queue and depending on the comparisons it rearranges the whole queue.
Complexity for this method is O (log n), n being the value count.

void swap (int i , int j): swaps two values in the given indexes i and j. Since set in the ArrayList class has the complexity O (1). This method has O(1) complexity also since it just uses the set method.

int compare (Color c1, Color c2):	compares two colors according to the comparator in the priority queue. Since all comparators' compare methods require O (1), this method has the time complexity of O (1) as well.
Color remove ():	removes the biggest color value in the queue and replaces the root of the heap in the queue with the right lower leaf of the heap and rearranges the whole heap with the required comparisons of the comparator in the priority queue. Complexity for this method is O (log n), n being the value count.
Color peek ():	Returns the biggest color value in the priority queue. Complexity is O (1) since it just returns the first element of the inner arraylist in the priority queue.
int getLeftChildIndex(int parentIndex):	Returns the left child's index by using the formula (parentIndex * 2) + 1 Complexity of this method is O (1).
int getRightChildIndex(int parentIndex):	Returns the right child's index by using the formula (parentIndex * 2) + 2 Complexity of this method is O (1).
int getParentIndex(int childIndex):	Returns the parent's value depending on what the child is. If the child's index is divisable by 2 then the parent index is

childIndex -2 / 2. If it is not, then the parent index is childIndex -1 /2.
The complexity of this method is O (1).

ImageProcessor Class: This class gets an image path and reads the image file. Then it gets pixels one by one, puts them into 3 different priority queues each having a unique comparator of: LEXComparator , EUCComparator or BMXComparator. This class also creates 4 threads one of them producing the pixels and the other 3 consuming them from the queues. This class have the following constructors and methods:

ImageProcessor(String imagePath):	creates an ImageProcessor instance and creates the 3 priority queues each having a unique comparator of : LEXComparator , EUCComparator or BMXComparator. After that this constructor reads the file which is denoted with the imagePath and stores it in a BufferedImage. Then it also creates 4 threads one of them producing the pixels using the producePixels method and the other 3 consuming them from the queues with the methods consumeLEX, consumeEUC and consumeBMX.
void producePixels():	gets the pixels from the BufferedImage one by one and creates color objects out of them. Then it adds the color to the 3 different priority queues. It also holds a count value to control when to start the other 3 threads. So when the count reaches 100 the other 3 thread starts. The complexity of this method is O(n). Since the first loop loops height and the inner loop

	loops width times their complexity is O(w*h) which is equal to the n = w*h, n being the pixel count.
void consumeLEX():	removes the pixels from the LEX priority queue and prints them out to the screen. Complexity of this function is O(w*h) which is equal to the n = w*h, n being the pixel count which is equal to O(n).
void consumeEUC():	removes the pixels from the EUC priority queue and prints them out to the screen. Complexity of this function is O(w*h) which is equal to the n = w*h, n being the pixel count which is equal to O(n).
void consumeBMX():	removes the pixels from the BMX priority queue and prints them out to the screen. Complexity of this function is O(w*h) which is equal to the n = w*h, n being the pixel count which is equal to O(n).

LEXComparator Class: This class implements Comparator interface to override its compare method to be used in the PriorityQueue class. It uses lexicographical comparison to do the comparisons of the two color objects. Class' only method is explained below:

int compare (Color c1, Color c2):	compares two colors with lexicographical comparison which is first comparing their red values, then their green values and then their blue values. If the first color is smaller then the second it returns -1, if they are equal it returns 0 and if the first color is bigger than the second color it returns 1. Complexity of this method is O (1).

EUCComparator Class: This class implements Comparator interface to override its compare method to be used in the PriorityQueue class. It uses Euclidean comparison to do the comparisons of the two color objects. Class' only method is explained below:

int compare (Color c1, Color c2):	compares two colors with Euclidean comparison which calculates their L2 norm which are the square root of the sum of all the values squared. If the first color is smaller than the second it returns -1, if they are equal it returns 0 and if the first color is bigger than the second color it returns 1. Complexity of this method is O (1).

BMXComparator Class: This class implements Comparator interface to override its compare method to be used in the PriorityQueue class. It uses Bitmix comparison to do the comparisons of the two color objects. Class' only method is explained below:

int compare (Color c1, Color c2):	compares two colors with Bitmix comparison which picks up the bits one by one from red, green and blue values of the color and creates a 24-bit long value. If the first color's 24-bit long value is smaller than the second it returns -1, if they are equal it returns 0 and if the first color's 24-bit long value is bigger than the second color it returns 1. Complexity of this method is O (1).

Color Class: This class represents colors which has red, green and blue values from 0-255. Class' constructors and methods are explained below:

Color (int red, int green ,int blue):	creates a color instance and initializes the red ,green ,blue values with the given red , green , blue values.

int getRed():	returns the red value of the color. Complexity of this method is O (1).
int getGreen ():	returns the green value of the color. Complexity of this method is O (1).
int getBlue():	returns the blue value of the color. Complexity of this method is O (1).

3 RESULT

3.1 Test Cases

Tested this program with multiple images which is provided below:

 blue.png  red.png  pattern.png

 green.png  gradient.png  example.png (the original size is much bigger)

These files will be provided in the project file for testing purposes.

3.2 Running Results

The running results are in the zip file in different pdf files.
For blue.png → blueOutput.pdf
For red.png → redOutput.pdf
For green.png → greenOutput.pdf
For pattern.png → patternOutput.pdf
For gradient.png → gradientOutput.pdf
Since the example.png's output is too big only the image is provided for testing purposes, not the resulting pdf file.
Depending on the thread scheduling of the OS the outputs may differ at each run. But the first thread always runs before the other 3 thread for sure no matter what. Also this thread scheduling may affect the order of the threads to run, first thread starts the other threads at the 100th pixel but still depending on the thread scheduling first thread may seem to run more than 100 times before the other threads.

- Main titles -> 16pt , 2 line break
- Subtitles -> 14pt, 1.5 line break
- Paragraph -> 12pt, 1.5 line break