Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 05 REPORT

Yusuf Can Kan
161044007

Course Assistant: Özgü Göksu

1 INTRODUCTION

1.1 Problem Definition

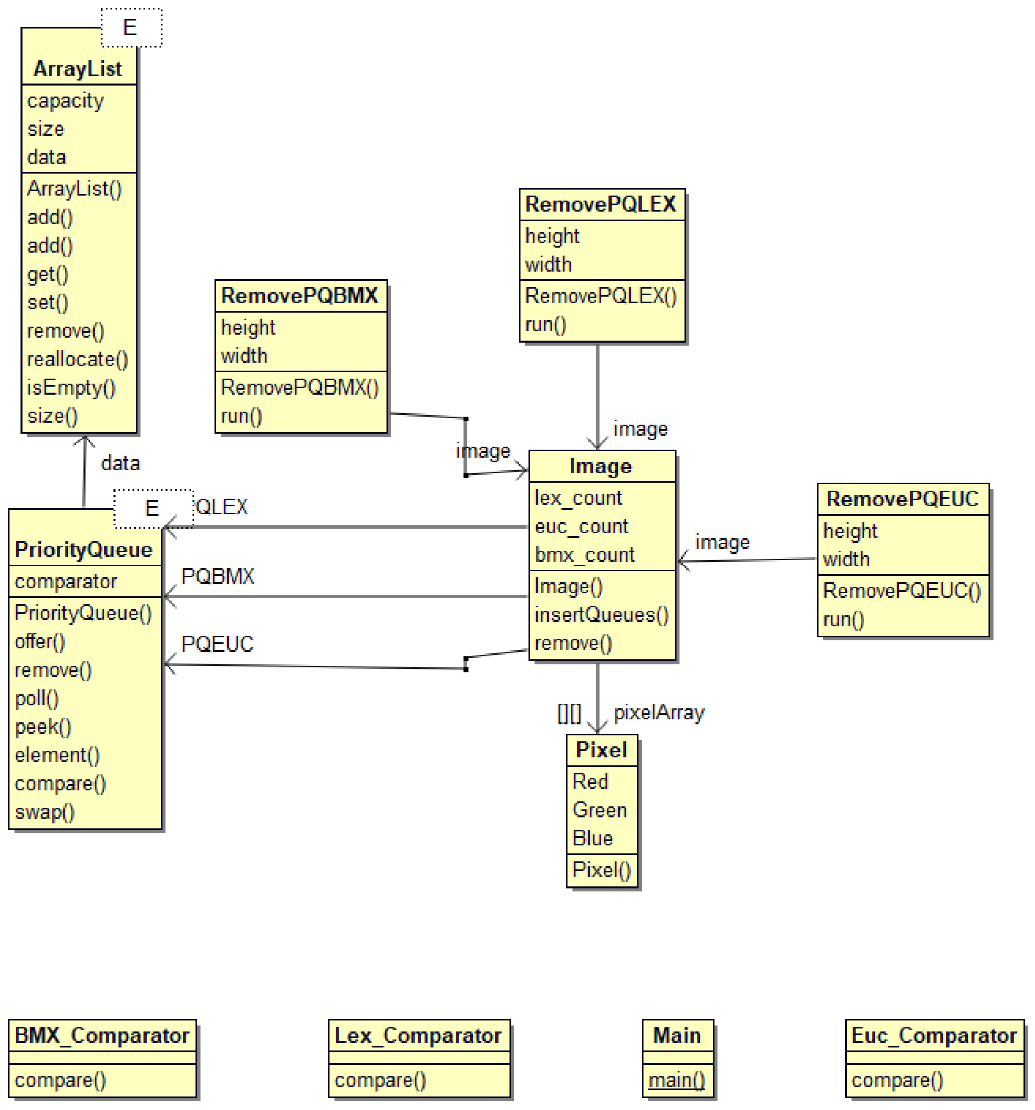
The problem which we want to solve is obtaining image data with respect to their pixels rgb values and constructing max priority queue with those pixels. When we adding the pixels to the queues we use three comparison techniques. Lexicographical comparison (LEX),Euclidean norm based comparison(EUC) and bitmix comparison(BMX). For every comparison technique we add the pixels to the different queues. The other things which is wanted to us using threads. First thread is adding thread, this thread adds element to 0 to 100 and after 100 this thread creates and starts the 3 threads which each of them removes 1 element from the priority queues. We have to work them properly,synchronized and efficiently.

1.2 System Requirements

My solution does not require spesific piece of hardware. It requires just a little memory and CPU(Because of the threads). You can use my solution with every computer unless the computer has (JVM). It works every machine which has JVM. It does not requires spesific operating system. You can use it on Linux operating system. My solution is so simple, you can use it almost everywhere.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams

This software can be use with simultaneous producing and consuming jobs and learning comparators with priority queue and learning simple threads problems(producer consumer problem). The expectations from user is giving a right input so user must give the jpg or png format picture. The only thing which user do is giving the path of picture with the arguement.

2.3 Problem Solution Approach

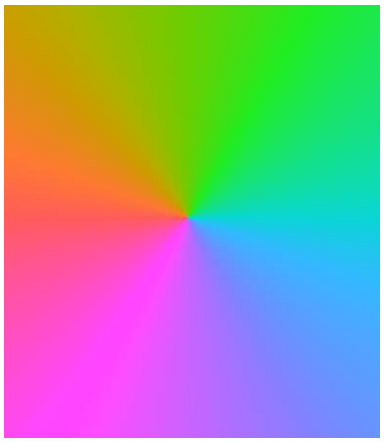
First i created a class for holding every pixels RGB values. After that i put the every pixel of the image into an two dimentional array. After that i implemented the comparators with different class and i putted pixels into the priority queues. After that i created first thread and tested it and after that i created other 3 thread. I didn't create different class for first thread, because its just adds the element but for the others i created for the simplicity of the code. I use ArrayList for holding data inside the priority queue because it is the most proper data structure for this design. When i designing threads, i paid attention to the producer consumer problem and for that i used wait() and notify() methods and to prevent the synchronization problem i used synchronized keyword. I made my design like that because it's so simple and efficient. I used priority queue and for priority queue i used array list. My time complexity is O(w*h) = O(n) which w is width pixel of picture and height is height pixels of picture. The space complexty is the same O(n).

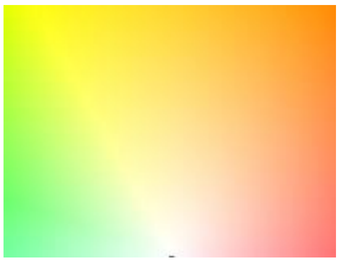
3 RESULT

3.1 Test Cases

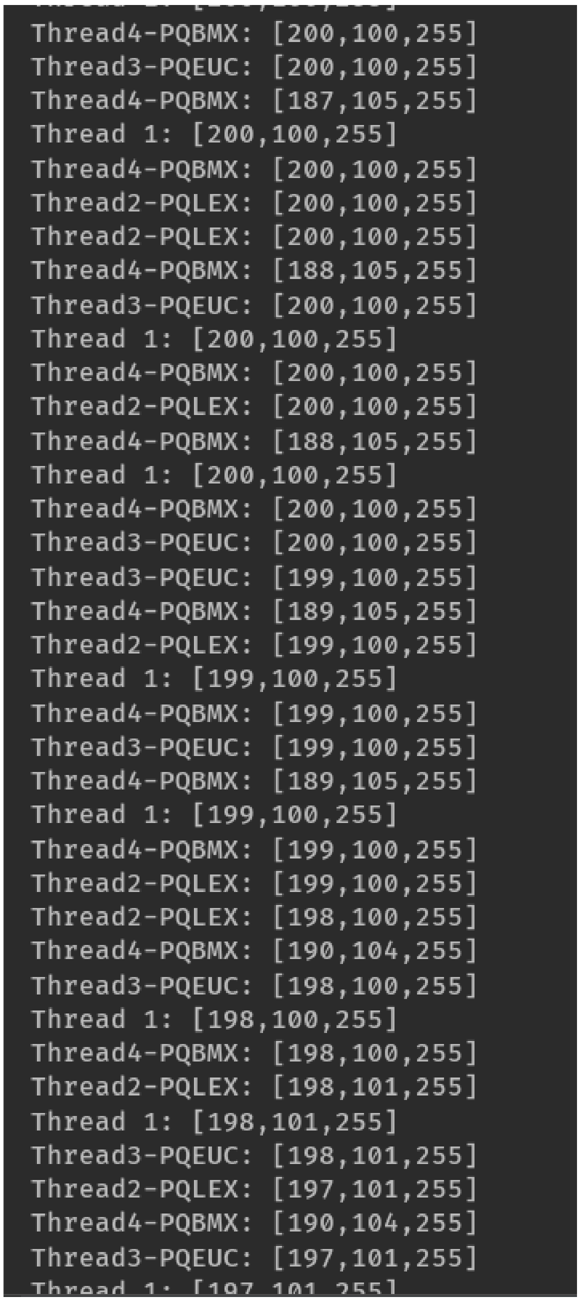
I tested it first part part, first tested with comparators with simple inputs, after that i tested priority queue. In the last i gave 2 pictures and i looked does all threads process every pixel or not.

Pictures;


Test 1

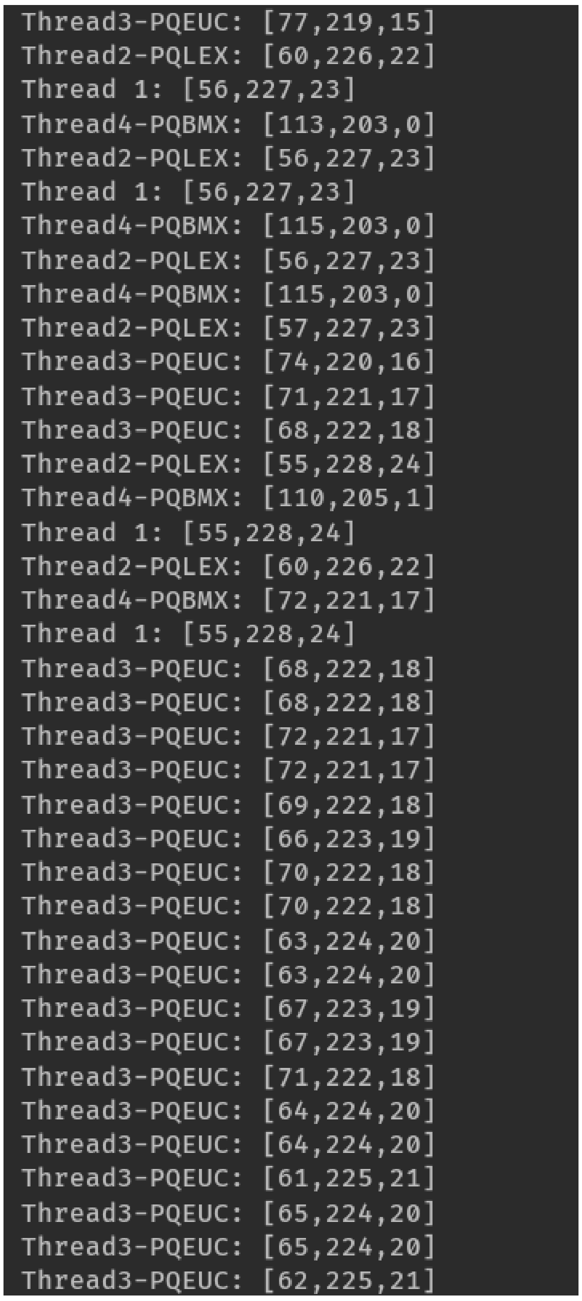

Test 2

3.2 Running Results



```
Thread4-PQBMX: [200,100,255]
Thread3-PQEUC: [200,100,255]
Thread4-PQBMX: [187,105,255]
Thread 1: [200,100,255]
Thread4-PQBMX: [200,100,255]
Thread2-PQLEX: [200,100,255]
Thread2-PQLEX: [200,100,255]
Thread4-PQBMX: [188,105,255]
Thread3-PQEUC: [200,100,255]
Thread 1: [200,100,255]
Thread4-PQBMX: [200,100,255]
Thread2-PQLEX: [200,100,255]
Thread4-PQBMX: [188,105,255]
Thread 1: [200,100,255]
Thread4-PQBMX: [200,100,255]
Thread3-PQEUC: [200,100,255]
Thread3-PQEUC: [199,100,255]
Thread4-PQBMX: [189,105,255]
Thread2-PQLEX: [199,100,255]
Thread 1: [199,100,255]
Thread4-PQBMX: [199,100,255]
Thread3-PQEUC: [199,100,255]
Thread4-PQBMX: [189,105,255]
Thread 1: [199,100,255]
Thread4-PQBMX: [199,100,255]
Thread2-PQLEX: [199,100,255]
Thread2-PQLEX: [198,100,255]
Thread4-PQBMX: [190,104,255]
Thread3-PQEUC: [198,100,255]
Thread 1: [198,100,255]
Thread4-PQBMX: [198,100,255]
Thread2-PQLEX: [198,101,255]
Thread 1: [198,101,255]
Thread3-PQEUC: [198,101,255]
Thread2-PQLEX: [197,101,255]
Thread4-PQBMX: [190,104,255]
Thread3-PQEUC: [197,101,255]
Thread 1: [197,101,255]
```

Test 1



```
Thread3-PQEUC: [77,219,15]
Thread2-PQLEX: [60,226,22]
Thread 1: [56,227,23]
Thread4-PQBMX: [113,203,0]
Thread2-PQLEX: [56,227,23]
Thread 1: [56,227,23]
Thread4-PQBMX: [115,203,0]
Thread2-PQLEX: [56,227,23]
Thread4-PQBMX: [115,203,0]
Thread2-PQLEX: [57,227,23]
Thread3-PQEUC: [74,220,16]
Thread3-PQEUC: [71,221,17]
Thread3-PQEUC: [68,222,18]
Thread2-PQLEX: [55,228,24]
Thread4-PQBMX: [110,205,1]
Thread 1: [55,228,24]
Thread2-PQLEX: [60,226,22]
Thread4-PQBMX: [72,221,17]
Thread 1: [55,228,24]
Thread3-PQEUC: [68,222,18]
Thread3-PQEUC: [68,222,18]
Thread3-PQEUC: [72,221,17]
Thread3-PQEUC: [72,221,17]
Thread3-PQEUC: [69,222,18]
Thread3-PQEUC: [66,223,19]
Thread3-PQEUC: [70,222,18]
Thread3-PQEUC: [70,222,18]
Thread3-PQEUC: [63,224,20]
Thread3-PQEUC: [63,224,20]
Thread3-PQEUC: [67,223,19]
Thread3-PQEUC: [67,223,19]
Thread3-PQEUC: [71,222,18]
Thread3-PQEUC: [64,224,20]
Thread3-PQEUC: [64,224,20]
Thread3-PQEUC: [61,225,21]
Thread3-PQEUC: [65,224,20]
Thread3-PQEUC: [65,224,20]
Thread3-PQEUC: [62,225,21]
```

Test 2