Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 5 REPORT

YUSUF PATOGLU
151044027

1 INTRODUCTION

1.1 Problem Definition

In this Project we'll deal with images with pixels. First we need to understand what are pixels and methods to compare them.Therefore we need to have a data structure that can efficiently store – retrieve values inside of it. Also another challange is work with multiple threads without keeping the CPU busy.

-What is an Image Pixel?

Image pixel contains three main colours these are red green and blue. All three have 8 bit values. So one pixel has 3D of 8 bit pixels.

-Why Binary Heap?

Instead of normal linear structure we can have a treelike structure to process our elements on logarithmic complexity.

-How to work with multiple threads?

Java's synchronized keyword configures the threads automaticly but we need to take extra cautions so we won't keep CPU busy.

1. System Requirements

If you want to output the terminal console to file you need to have space like 41 MB in order to get all values. It consumes a lot RAM like 1 Gigabyte, so default JVM settings should be satisfied..

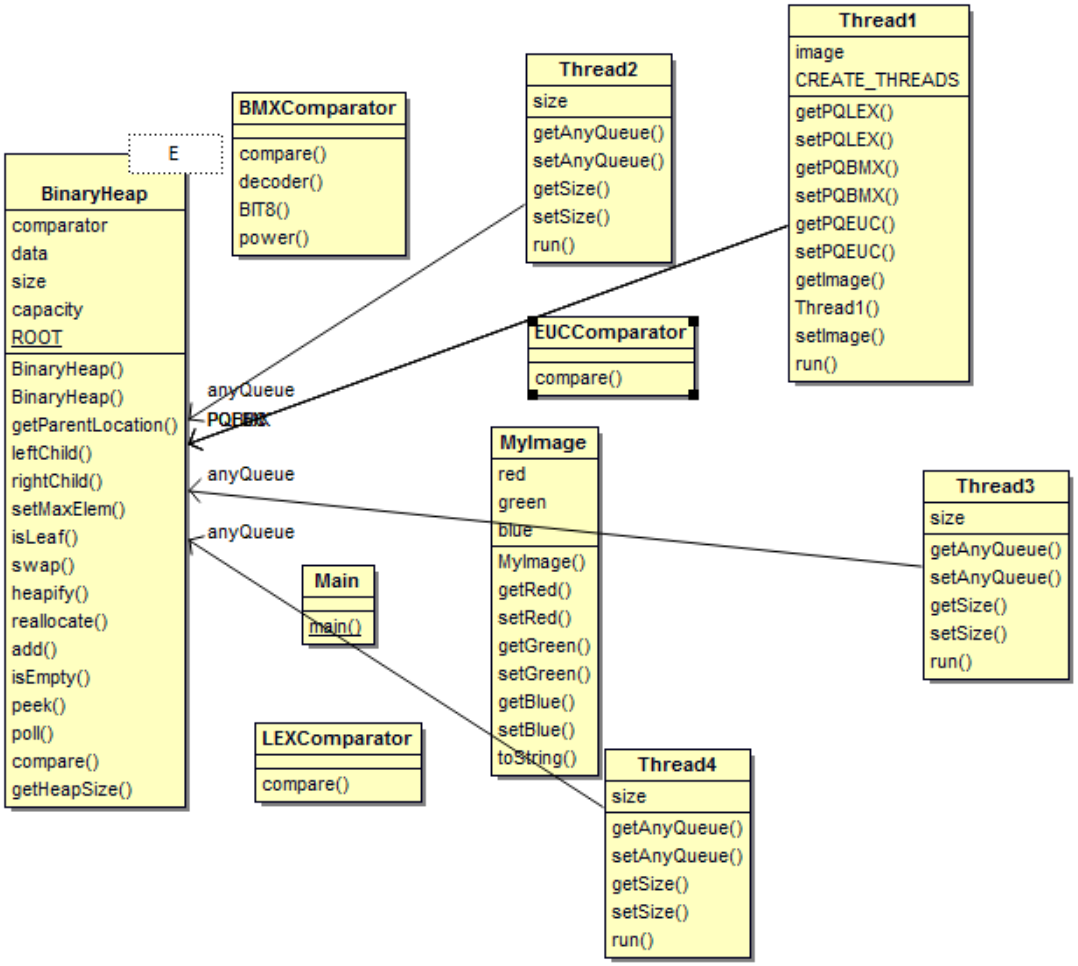
Just in case JVM system requirements are noted here on Java's website:

The minimum and recommended memory and disk space requirements are as follows:

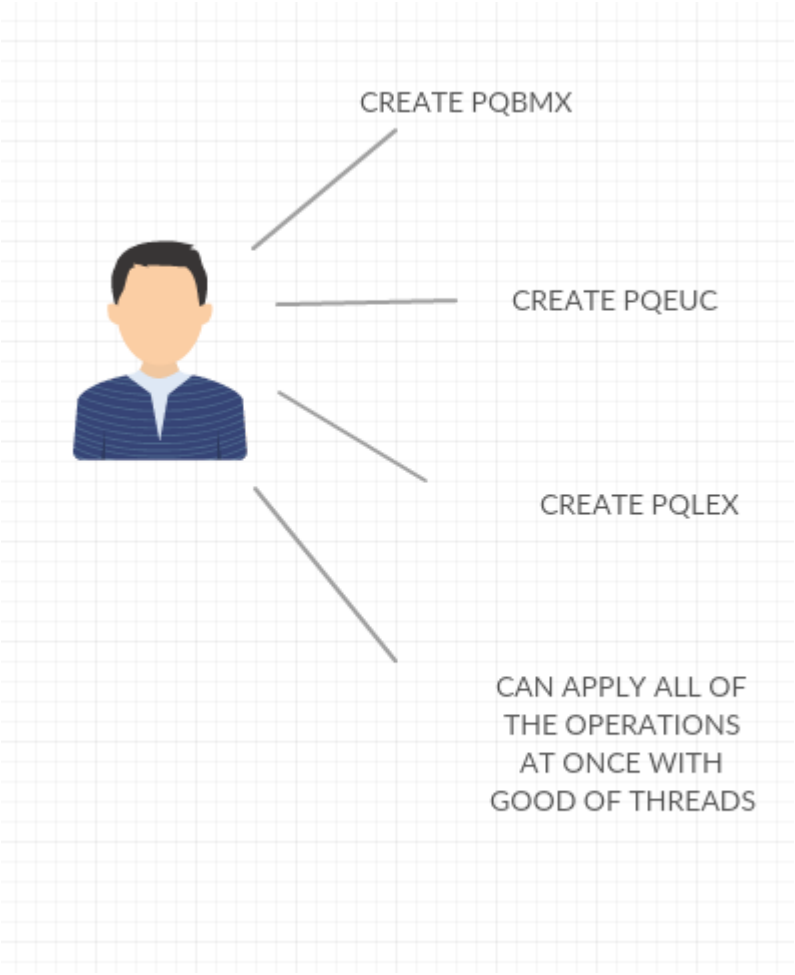
- • Minimum memory: 1 GB
- • Recommended memory: 2 GB for Windows platforms, 1 GB for non-Windows platforms
- • Minimum disk space: 250 MB
- • Recommended disk space: 500 MB

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagram



2.3 Problem Solution Approach

- In order to store my elements I used binary heap. Actually my intention was implementing a Fibonacci or Binomial heap to get extra efficiency. In theory I understand them but since my knowledge is limited I couldn't implement those data structures. Therefore I used classical binary heap. Since I worked with tree like binary heap class the efficiency is always logarithmic.

- I created three comparator classes these are defined in PDF. For the sake of OOP I only used one binary heap implementation and reached that three methods through this binary heap class.
- Also I had a Image class to keep color properties for the sake of clarification.

- I had one thread class to create and start other tree. First I couldn't find any solutions without sleep() but after that I realised simple mathematical solution is the key to implement this logic without sleep(). (Calculated via width* height)

3 RESULT

3.1 Test Cases

When I implemented my binary heap class first I tried with simple integer values since it was a generic class and saw if I implemented it correct or not. Then I used myImage class as heap data.

I see there was no problems with that too. Then I implemented three comparators and tested them with unique pixel values to see I missed something or not.

When it comes to testing the heap with an image pixels, stdout is not enough. I decided to write these properties of pixels to another file. I will show them in running results section.

3.2 Running Results

This is the first few lines of PQLEX Heap(Max Values):

```
*********EXTRACTING PQLEX IMAG
Red: 255 Green: 122 Blue: 51
Red: 255 Green: 122 Blue: 51
Red: 255 Green: 122 Blue: 51
Red: 255 Green: 122 Blue: 50
Red: 255 Green: 122 Blue: 50
Red: 255 Green: 122 Blue: 50
Red: 255 Green: 122 Blue: 50
Red: 255 Green: 122 Blue: 50
Red: 255 Green: 122 Blue: 50
Red: 255 Green: 122 Blue: 50
Red: 255 Green: 122 Blue: 50
Red: 255 Green: 121 Blue: 52
Red: 255 Green: 121 Blue: 52
Red: 255 Green: 121 Blue: 52
Red: 255 Green: 121 Blue: 52
Red: 255 Green: 121 Blue: 52
Red: 255 Green: 121 Blue: 52
Red: 255 Green: 121 Blue: 52
```

This is the first few lines of PQEUC Heap(Max Values):

```
*********EXTRACTING PQEUC IMAGE
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
Red: 255 Green: 77 Blue: 255
```

This is the size of terminal output(With all threads):

C:\CodeRep\ImagePriority

40,9 MB (42.907.288 bytes)

40,9 MB (42.909.696 bytes)

For first 100 elements it contains:

```
 89   Thread1 : 218, 147, 14
 90   Thread1 : 218, 147, 14
 91   Thread1 : 218, 147, 14
 92   Thread1 : 219, 146, 15
 93   Thread1 : 219, 146, 15
 94   Thread1 : 219, 146, 15
 95   Thread1 : 220, 146, 16
 96   Thread1 : 220, 146, 16
 97   Thread1 : 220, 146, 16
 98   Thread1 : 220, 146, 16
 99   Thread1 : 220, 145, 16
100   Thread1 : 220, 145, 16
```

After that other threads starts to execute:

```
Thread2-PQLEX:[220, 146, 16]
Thread4-PQBMX:[220, 146, 16]
Thread3-PQEUC:[220, 146, 16]
Thread1 : 220, 145, 16
Thread3-PQEUC:[220, 146, 16]
Thread4-PQBMX:[220, 146, 16]
Thread2-PQLEX:[220, 146, 16]
Thread4-PQBMX:[220, 146, 16]
Thread3-PQEUC:[221, 145, 17]
Thread1 : 221, 145, 17
Thread3-PQEUC:[220, 146, 16]
Thread4-PQBMX:[220, 146, 16]
Thread2-PQLEX:[221, 145, 17]
Thread4-PQBMX:[221, 145, 17]
Thread3-PQEUC:[221, 145, 17]
Thread1 : 221, 145, 17
```