04.01.2020

FALL 2019- ASSİGNMENT3

CMPE150.06

HATİCE ERK-2018400090

PROBLEM DESCRIPTION

In this assignment, writing a program which makes some updates on some ppm images is expected. This program consists of 4 parts. First one is just about reading a ppm file and writing another ppm file. Secondly, we should get a black-and-white version of an image by some calculations. Then, third one is convolution part which takes a filter and operate it. In the last one, the program should take an integer as the range and make color quantization.

PROBLEM SOLUTION

In this assignment, I used 5 method which one is recursion. First, I described 'mode' and took my input. 'imageFormat', 'rows', 'columns' and 'max' are described. Then due to the 'mode' value, my code operates parts.

In part one, my method 'writingPPMFile' takes an array, an imageFormat and a max value. Rows and columns will be described by arrays length. It creates a ppm file 'output.pmm'. It writes imageFormat, number of rows and columns, max value in turn. Then it starts writing pixels.

In part two, my method 'blackAndWhite' takes again an array, an imageFormat and a max value. The aim of this part is getting a black-and-white version of image. To get black and white, my code takes average of pixel values and all pixel values equals it. Then similar to part one, it writes another ppm file 'black-and-white.ppm'.

In part three, my method 'filter' takes an array, an imageFormat, a max value and also a filter array. The aim of this part is filtering the image by given filter. Before this

method, in main part my code takes a txt file and changes it an array. After some calculations, it creates a new array arrayNew and storage results of calculations to this array. Then similar to part one, it writes another ppm file 'convolution.ppm'.

In last part, I used two method 'quantization' and 'recursionQuan'. Before this method, in main part, I took a range value and the method 'quantization' takes an array, an imageFormat, a max value and this range value. My code traverse the pixels of an input image one by one, check if the values of a pixel's neighbors are within a given range, and make their values equal if they are in the same value range. To check this I used the method 'recursionQuan'. After checking, similar to part one, it writes another ppm file 'quantizied.ppm'.

IMPLEMENTATION

```
1.  import java.util.*;
2.  import java.io.*;
3.  public class HE2018400090 {
4.      public static void main(String[] args) throws FileNotFoundException{
5.
6.          int mode = Integer.parseInt(args[0]);
7.          //The variable 'mode' will show which part should be operated.
8.
9.          File f = new File(args[1]);
10.         //The variable 'f' is the file which is input.
11.         Scanner file = new Scanner(f);
12.         //I use scanner here to get format and integers easily.
13.
14.         String imageFormat = file.nextLine();
15.         //'imageFormat' represents 'P3' everywhere in my code.
16.
17.         int rows = file.nextInt();
18.         int columns = file.nextInt();
19.         //'rows' and 'columns' shows how much column and row ppm file has.
20.
21.         int max = file.nextInt();
22.         //'max' is the maximum value of a pixel.
23.
24.         //Here, I read the ppm file and create an 3D array.
25.         int[][][] imageArray = new int[rows][columns][3];
26.         for(int e=0; e<rows; e++){
27.             for(int r=0; r<columns; r++){
28.                 for(int k=0; k<3; k++){
29.                     imageArray[e][r][k]= file.nextInt();
30.                 }
31.             }
32.         }
33.
34.         //Now, due to the value of 'mode', it's operation time.
35.         if(mode==0){
36.             //when 'mode' is zero,code just does the writing another file.
37.             writingPPMFile(imageFormat,max,imageArray);
```

```
        }if(mode==1){
            //when 'mode' is one, code gives a black-and-white version of image.
            blackAndWhite(imageArray, imageFormat, max);
        }if(mode==2){
            //when 'mode' is two, code should operate a filter on image.

            //before the method, first I changed filter to a 2D array.
            File filter2 = new File(args[2]);
            Scanner file2 = new Scanner(filter2);

            String size = file2.nextLine();
            int a = Integer.parseInt(size.substring(0,size.indexOf("x")));
            int b = Integer.parseInt(size.substring(size.indexOf("x")+1));

            int[][] filter = new int[a][b];
            for(int h=0; h<a; h++){
                for(int e=0; e<b; e++){
                    filter[h][e] = file2.nextInt();
                }
            }
            //Then,method time.
            filter(imageArray,max,filter,imageFormat);
        }if(mode==3){
            //when 'mode' is three, code should quantize the image.
            //First, I took the range.
            int range = Integer.parseInt(args[2]);
            //Then, the method.
            quantization(imageArray,range,imageFormat,max);
        }

    }

    //To write pixels to another file.
    public static void writingPPMFile(String imageFormat, int max, int[][][] array)t
hrows FileNotFoundException {
        //First, to create a new file, we should throw FileNotFoundException.

        //Here, I create the output file.
        File f = new File("output.ppm");
        PrintStream output = new PrintStream(f);

        int rows = array.length;
        int columns = array[0].length;

        //First I write to the file the necessary things.
        output.println(imageFormat);
        output.println(rows  + " " + columns);
        output.println(max);
        //then, I wrote the value of pixels.
        for(int e=0; e<rows; e++ ){
            for(int r=0; r<columns; r++){
                for(int k=0; k<3; k++){
                    output.print(array[e][r][k]+ " ");
                }
                output.print("\t");
            }
        }

    }

    //To get a black-and-version of image.
    public static void blackAndWhite(int[][][] array, String imageFormat, int max)th
rows FileNotFoundException{
        //First, to create a new file, we should throw FileNotFoundException.

```

```
101.                //To get a black-and-
    white version, we should average the pixel values.
102.                for(int e=0; e<array.length; e++){
103.                    for(int r=0; r<array[0].length; r++){
104.                        int sum =0;
105.                        for(int k=0; k<3; k++){
106.                            sum += array[e][r][k];
107.                        }
108.                        for(int k=0; k<3; k++){
109.                            array[e][r][k]=sum/3;
110.                        }
111.                    }
112.                }
113.                //Now, in the array pixel values like 'x,x,x'.
114.
115.                // Rest of method is the same as writingPPMFile method.
116.                File f = new File("black-and-white.ppm");
117.                PrintStream output = new PrintStream(f);
118.
119.                int rows = array.length;
120.                int columns = array[0].length;
121.
122.                output.println(imageFormat);
123.                output.println(rows  + " " + columns);
124.                output.println(max);
125.
126.                for(int e=0; e<rows; e++ ){
127.                    for(int r=0; r<columns; r++){
128.                        for(int k=0; k<3; k++){
129.                            output.print(array[e][r][k]+ " ");
130.                        }
131.                        output.print("\t");
132.                    }
133.                }
134.            }
135.            //To operate filter on image.
136.            public static void filter(int[][][] array,int max, int[][] filter, String
    imageFormat)throws FileNotFoundException{
137.                //First, to create a new file, we should throw FileNotFoundException.

138.
139.                int a = filter.length;
140.                int b = filter[0].length;
141.
142.                int[][][] arrayNew = new int[array.length-2][array[0].length-2][3];
143.                //I create a new array to storage results of calculations. It is smal
    ler than first array.
144.
145.                for(int e=0; e<array.length-a+1; e++){
146.                    for(int r=0; r<array[0].length-b+1; r++){
147.                        for(int k=0; k<3; k++){
148.                            int sum=0;
149.                            for(int h=0; h<a; h++){
150.                                for(int t=0; t<b; t++){
151.                                    sum += filter[h][t]*array[e+h][r+t][k];
152.                                }
153.                            }
154.                            if(sum/a*b<0){
155.                                sum=0;
156.                            }if(sum/a*b>max){
157.                                sum=max;
158.                            }
159.                            arrayNew[e][r][k] = sum/a*b;
160.                        }
161.                    }
162.                }
```

```
163.
164.                 // now, arrayNew should be black-and-
    white. This part is same as black-and-white method.
165.                 for(int e=0; e<arrayNew.length; e++){
166.                     for(int r=0; r<arrayNew[0].length; r++){
167.                         int sum =0;
168.                         for(int k=0; k<3; k++){
169.                             sum += arrayNew[e][r][k];
170.                         }
171.                         for(int k=0; k<3; k++){
172.                             arrayNew[e][r][k]=sum/3;
173.                         }
174.                     }
175.                 }
176.                 //This part is the same as writingPPMFile method.
177.                 File f = new File("convolution.ppm");
178.                 PrintStream output = new PrintStream(f);
179.
180.                 int rows = arrayNew.length;
181.                 int columns = arrayNew[0].length;
182.
183.                 output.println(imageFormat);
184.                 output.println(rows  + " " + columns);
185.                 output.println(max);
186.
187.                 for(int e=0; e<rows; e++ ){
188.                     for(int r=0; r<columns; r++){
189.                         for(int k=0; k<3; k++){
190.                             output.print(arrayNew[e][r][k]+ " ");
191.                         }
192.                         output.print("\t");
193.                     }
194.                 }
195.             }
196.             //To quantize the image.
197.             public static void quantization(int[][][] array,int range, String imageFo
    rmat, int max)throws FileNotFoundException{
198.                 //First, to create a new file, we should throw FileNotFoundException.

199.
200.                 //I looked every pixel to look every neighbor.
201.                 for(int k=0; k<3; k++){
202.                     for(int r=0; r<array[0].length; r++){
203.                         for(int e=0; e<array.length; e++){
204.                             int value = array[e][r][k];
205.                             recursionQuan(array, value, range, e, r, k);
206.                         }
207.                     }
208.                 }
209.                 //This part is the same as writingPPMFile method.
210.                 File f = new File("quantized.ppm");
211.                 PrintStream output = new PrintStream(f);
212.
213.                 int rows = array.length;
214.                 int columns = array[0].length;
215.
216.                 output.println(imageFormat);
217.                 output.println(rows  + " " + columns);
218.                 output.println(max);
219.
220.                 for(int e=0; e<rows; e++ ){
221.                     for(int r=0; r<columns; r++){
222.                         for(int k=0; k<3; k++){
223.                             output.print(array[e][r][k]+ " ");
224.                         }
225.                         output.print("\t");
```

```
                    }
                }

            }
            //To look every neighbor of pixel.
            public static int[][][] recursionQuan(int[][][] array, int value, int ran
    ge, int e, int r, int k){
                if(e<array.length-1){
                    //I looked the right pixel.
                    if(array[e+1][r][k]<=value+range && value-
    range<=array[e+1][r][k] && !(array[e+1][r][k]==value)){
                        array[e+1][r][k] = value;
                        recursionQuan(array, value, range, e+1, r, k);
                    }
                }

                if(e>0){
                    //I looked the left pixel.
                    if(array[e-1][r][k]<=value+range && value-range<=array[e-
    1][r][k] && !(array[e-1][r][k]==value)){
                        array[e-1][r][k] = value;
                        recursionQuan(array, value, range, e-1, r, k);
                    }
                }

                if(r<array[0].length-1){
                    //I looked the under.
                    if(array[e][r+1][k]<=value+range && value-
    range<=array[e][r+1][k] &&  !(array[e][r+1][k]==value)){
                        array[e][r+1][k] = value;
                        recursionQuan(array, value, range, e, r+1, k);
                    }
                }
                if(r>0){
                    //I looked the top.
                    if(array[e][r-1][k]<=value+range && value-range<=array[e][r-
    1][k] && !(array[e][r-1][k]==value)){
                        array[e][r-1][k] = value;
                        recursionQuan(array, value, range, e, r-1, k);
                    }
                }

                if(k<2){
                    //I looked inside.
                    if(array[e][r][k+1]<=value+range && value-
    range<=array[e][r][k+1] && !(array[e][r][k+1]==value)){
                        array[e][r][k+1] = value;
                        recursionQuan(array, value, range, e, r, k+1);
                    }
                }
                if(k>0){
                    //I looked outside.
                    if(array[e][r][k-1]<=value+range && value-range<=array[e][r][k-
    1] && !(array[e][r][k-1]==value)){
                        array[e][r][k-1] = value;
                        recursionQuan(array, value, range, e, r, k-1);
                    }
                }

                return array;
            }
        }
```

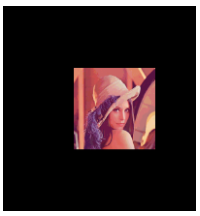
OUTPUT OF THE PROGRAM

mode 0

mode 1

mode 2 – vertical

mode 3 - 8



CONCLUSION

In this assignment, I did first three part successfully. In last part, I think my code is true but I had some differences between images. Because I didn't understand why they differs, I am not so happy. However I learned about ppm images and how I can change an image by filter, which makes me more proud.