Project 7.1: You are to implement those four (4) RunLength Encoding methods taught in class:

Method 1) Encode without zero and no wrap-around.

Method 2) Encode without zero and wrap-around.

Method 3) Encode with zero and no wrap-around. //algorithm given below.

Method 4) Encode with zero and wrap-around.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Language: Java

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Due date: Soft copies: 4/11/2018, Wednesday before Midnight

Early submission deadline: 4/8/2018, before midnight

Due date: Hard copies: 4/12/2018, Thursday in class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I. Inputs:

1) Input (argv): An image file in txt format

2) Console: Ask user for input

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

II. Outputs:

1) Console: Display the four methods on the console as given in the above;

then ask the user to enter (1, 2, 3, or 4) for the method choice

2) Output (argv): the result of the encoded image, in the given format below.

\*\*\* Format of the encoded run-length:

The first text line is the input image header;

The second text line is the method used;

follows by one run-length per text line

Example:

20 15 0 9 // header information

2 // method 2 was used to encode

1 4 8 10 // startRow is 1, startCol is 4, color is 8, 10 pixels long

2 4 7 5 // startRow is 2, startCol is 4, color is 7, 5 pixels long

:

:

:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

III. Data structure: On your own!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Use object-oriented data structures similar to those given in the previous project specs.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IV. Algorithm in main

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

step 0: - open all files

* initializing those needed in the data structures
* Read the image header and output the image header to outFile

step 1: Display to the console those four method and ask the user to input choice

step 2: method <-- get from the user from the console

if method is not within 1 – 4

exit with error message

step 3: output method to outFile

case of method

1: call method1 (inFile) // given in class

2: call method2 (inFile) // on your own

3: call method3 (inFile) // see algorithm method3 below

4: call method4 (inFile) // on your own

step 4: closed all files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Algorithm steps for Method 3

// Encode with zero and no wrap-around.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Step 0: r <-- 0

Step 1: Scan inFile left to right and top to bottom

c <-- 0

count <-- 0

currVal <-- p (r, c)

Step 2: output r and c and currVal to outFile

count ++

Step 3: c++

Step 4: nextVal <-- p (r, c) // read the next pixel

Step 5: if nextVal == currVal

count++

else

output count to outFile

currVal <-- nextVal

count <-- 1

output r and c and currVal

Step 6: repeat step 3 to step 5 until end of text line

Step 7: r++

Step 8: repeat Step 1 to Step 7 until end of file