Student: ANCA STEFAN GABRIEL

Grupa: 333AA

Curs: Aplicatii Web cu Suport Java

Facultatea de Automatica si Calculatoare, 2020

TEMA DE PROCESARE

CONVERTING COLOR IMAGE TO GRAY-SCALE IMAGE – AVERAGE METHOD

Introducere

Majoritatea telefoanelor mobile dispun de o camera colora, foarte putine dintre acestea dispun si de una monocroma. Astfel ca pe fotografiile rezultate in format color, RGB, se pot aplica diferiti algoritmi pentru convertirea acestor folografii din RGB in alb-negru (grayscale). Astefel, pentru telefoanele mobile care nu dispun de o camera monocroma, putem integra o functionalitate (in java pentru android) care va rula in spate un algoritm special.

Descrierea aplicatiei cerute

Avand la dispozitie imagini colore, RGB, in format bitmap si cu 24 bit depth, vom aplica concepte de POO pentru a crea o aplicatie modulara cu 3 niveluri de mostenire, pentru a citi imaginea, pentru a verifica daca aceasta indeplineste cerintele, adica fiind in format BMP si avand 24 bit depth, vom citi imaginea, o vom prelucra prin aplicarea algoritmului AVERAGE, adica fiecare pixel avand canalele R, G, B, pixelul va fi inlocuit cu media celor 3 valori. Dupa ce imaginea va fi convertita la Grayscale, vom stoca imaginea tot in formatul BMP.

Partea teoretica

Formatul BMP (sau Bitmap Image (harta de biti)) este un format de fisiere imagine folosit pentru stocarea imaginilor independente de dispozitivul de afisare. Formatul bmp este capabil sa stocheze imagini bidimensionale, de inaltime si latime mari, monocrome sau color. Imaginea in format 24 bit semnifica utilizarea a 3 bytes pentru fiecare pixel, adica valorile pentru Rosu, Verde, Albastru vom fi stocare fiecare pe cate 8 biti, rezultand imagini de culori cu o acuratete extrem de mare.

Metoda folosita pentru convertirea imaginei RGB catre Grayscale este metoda mediei. Si anume, pentru fiecare pixel din imagine vom extrage valorile pe 8 biti pentru Rosu, Verde si Albastru si vom face media acestor valori pentru a obtine un pixel “gri”. Aceasta metoda nu este cea mai potrivita pentru convertirea imaginilor, deoarece in functie de tipul culorilor folosite in imagini, ne pot rezulta imagini foarte intunecate sau foarte luminoare, continutul nemaifiind foarte vizibil. Insa, pentru conversii rapide ale imaginilor RGB, putem folosi aceasta metoda.

Descrierea implementarii

Sa luam ca punct de plecarea modul cum utilizatorul va interactiona cu aplicatia, aceasta fiind apelata din linia de comanda.

Avand fisierele sursa ale aplicatiei, modul de executarea al aplicatiei este urmatorul:

- deschidem o consola si navigam in folderul **AncaStefanAWJ\out\production\ColoredImageToGrayscale**
- aici vom folosi comanda **java convert.ColoredImageToGrayscale** si drept argumente avem: pathtofileorfolder (required) si al doilea (obtional) este **-m** pentru a specifica rularea multithreaded. Cel mai simplu putem folosi

```
C:\Users\Stefan\Desktop\AncaStefanAWJ\out\production\ColoredImageToGrayscale>java convert.ColoredImageToGrayscale -help
The argument list should look like this:

pathToFileOrFolder [-m]

You have to provide the path to a file or folder
and if you provide a folder, you can specify if
you want to process the files multithreaded -m.

C:\Users\Stefan\Desktop\AncaStefanAWJ\out\production\ColoredImageToGrayscale>
```

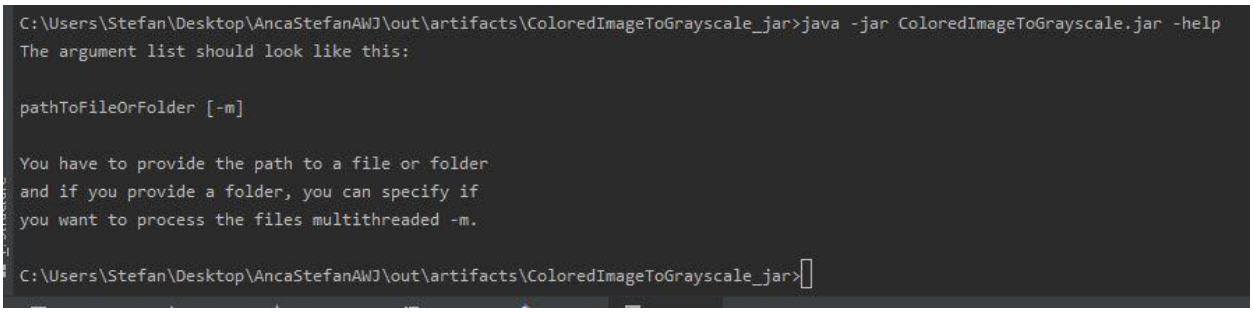
comanda **java convert.ColoredImageToGrayscale -help** care ne va afisa:

- insa userul va avea acces, cel mai probabil, la fisierul generat in format .jar, unde sintaxa este asemanatoare:

Structura aplicatiei

Avem clasa principala **ColoredImageToGrayscale** care contine metoda **main**

Avem interfata **DisplayElapsedTime** care contine metodele:



- **public void printExecutionTime()** care va fi mostenita de alte clase si va printa timpul de executie pentru fiecare tip de procesare (citire, convertire si scriere)
- **public void setStarting(long starting); public long getStarting(); public void setEnding(long ending); public long getEnding();** cu ajutorul carora vom retine si seta timpul de inceput, respectiv timpul de finalizare al procesarii, iar prin diferenta dintre aceste doua valori vom obtine timpul de procesare
- **public void setSuccess(boolean processSuccess)** cu ajutorul acestei metode vom seta starea de succes sau nu a executiei procesarii, avem nevoie de acest flag pentru a sti daca vom continua sau nu procesare imaginii
- **public boolean getSuccess()** va returna valoarea flagului (true, procesarea etapei s-a realizat cu succes, sau false, in timpul procesarii a aparut o eroare

Clasa abstracta **StoreTime** implementeaza interfata **DisplayElapsedTime**, contine atributul **processSuccess** pentru care vom suprascrie getter (getSuccess) si setter (setSuccess). Avem, de asemenea, alte doua atribute (**startingTime, endingTime**), alaturi de getter si setter, care vor contine timpul de incepul al procesarii, respectiv timpul de finalizare. Metodele abstracte **process()** si

returnImage() vor fi suprascrise si se vor ocupa de procesarea imaginilor, respectiv de returnarea imaginilor ca **BufferedImage**.

Vom avea clase separate pentru citirea imaginii (**ReadImage**), convertirea acesteia din RGB la Grayscale (**ConvertToGrayscale**) si scrirea noii imagini inapoi pe disk (**WriteImage**). Aceste clase extind clasa abstracta.

Pentru citirea imaginii vom avea nevoie de path-ul acesteia. Avand locatia imaginii, vom verifica daca aceasta este valida, vom verifica daca fisierul este in format BMP, iar apoi vom folosi **ImageIO.read** pentru citirea imaginii de pe disk si o vom stoca ca **BufferedImage**. Daca citirea se executa cu success, vom seta flagul ca **true** si vom afisa timpul total pentru citirea imaginii.

Pentru convertirea imaginii va trebui sa preluam dimensiunile acesteia, iar apoi pentru fiecare pixel (folosind doua for-uri) vom extrage valorile pentru R, G, B si vom face media acestora, urmand sa setam din nou valoarea pentru acel pixel cu media tocmai aflata. Daca procesarea s-a realizat cu success, vom seta flagul pe **true**.

Pentru scrierea imaginii convertite, avem nevoie de locatia unde vom scrie imaginea.

MainClass este clasa care implementeaza **Runnable**, si va trebui sa suprascriem metoda **run()**. In aceasta metoda vom crea obiecte pentru cele 3 clase (**ReadImage, ConvertToGrayscale, WriteImage**). Daca path-ul este valod, acesta va fi parsat obiectului pentru citirea imaginii, iar daca **getSuccess** returneaza true (executie fara probleme), vom parsa imaginea citita obiectului pentru convertirea imaginii, acesta ne va returna imaginea convertita. Vom crea un nou folder denumit “Grayscale” (in cazul in care acesta nu exista deja), unde vom stoca imaginile care vor avea adaugat la denumirea lor “_grayscale”. Daca si scrirea se realizeaza cu succes, vom printa timpul total in care s-a realizat citirea, convertirea si scrierea imaginii.

Pentru metoda **main** avem nevoie de o alta metoda care ne va decodifica argumentele trimise in linia de comanda. Aceasta metoda se gaseste in clasa principala (**ColoredImageToGrayscale**) si va vedea daca nu avem niciun argument, ne vom intoarece in main cu **false** si vom intrerupe rularea aplicatiei pentru argumente invalide. Daca avem un argument, acesta trebuie sa fie calea catre

fisierul sau folderul nostru. Verificam daca aceasta cale este un fisier (si vom seta atributul **isFile** ca true) sau daca este un folder (setam **isFolder** pe true), iar in caz contrar ne vom intoarce in main cu false.

Verificam daca primul argument este **-help**, atunci vom afisa utilizatorului instructiunile necesare rularii aplicatiei (pozele prezentate anterior). Pentru doua argumente vom face aceleasi verificari pentru primul ca mai sus, iar pentru al doilea verificam daca este **-m**, atunci utilizatorul a ales prelucrarea multithreaded si vom seta **multithreading** pe true.

Ne intoarcem in main, unde verificam daca decodificarea returneaza true, insemna ca argumentele sunt valide si incepem instantierea de obiecte pentru procesare. Daca calea este una catre un fisier, vom crea un nou obiect temporar pentru clasa **MainClass** pentru care vom apela metoda **run()** si vom prelucra fisierul. Toate verificarile sunt efectuate in clasele prezentate mai sus.

Daca calea este una catre un folder, vom crea un array de String-uri in care vom stoca toate denumirile fisierelor. Daca acest array nu contine niciun element, inseamna ca folderul este gol si vom opri aplicatia cu un mesaj corespunzator. Daca avem elemente, verificam daca a fost selectata optiunea pentru multithreading si vom crea cate un thread pentru fiecare fisier de procesat. Nu cred ca este cazul ca limitam numarul de threaduri deoarece presupunem ca aceasta metoda de procesare este folosita pentru un numar mic de fisiere. Daca optiunea de multithreading nu este selectata, vom crea obiecte temporare si vom face prelucrarea secventiala (se poate vedea diferenta dintre timpii de executie pentru metoda **main**.

Performante

Rezultatele prelucrarii cu aceasta metoda sunt unele chiar bune. Implementarea pare a fi una buna, toate exceptiile fiind tratate. Performantele pentru procesare par a fi bune, insa nu avem o alta metoda implementata pentru compararea rezultatelor.

Bibliografie

- http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/
- https://etc.usf.edu/techease/win/images/what-is-bit-depth/
- https://itnext.io/bits-to-bitmaps-a-simple-walkthrough-of-bmp-image-format-765dc6857393
- https://www.tutorialspoint.com/java_dip/java_buffered_image.htm
- https://www.w3schools.com/java/java_abstract.asp
- https://www.tutorialspoint.com/java/java_object_classes.htm
- https://www.codejava.net/java-se/file-io/how-to-list-files-and-directories-in-a-directory
- https://www.geeksforgeeks.org/multithreading-in-java/
- https://stackoverflow.com/questions/53379634/java-multi-threading-for-text-file-processing
- https://www.callicoder.com/java-multithreading-thread-and-runnable-tutorial/