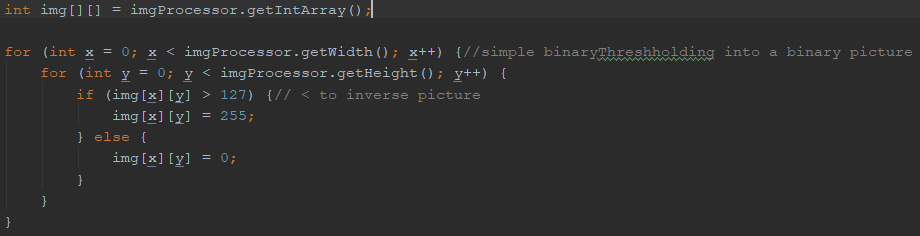
Sean McGrath CSC474 – image processing

I decided to just make a backend Java program that grabs a picture in a specified directory and returns a connected components image to that directory.

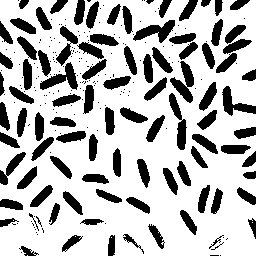
The picture needs to already be in greyscale format and a JPG or PNG, a grey RGB image for example wouldn’t work for this program. A binary image would only work if it’s a greyscale format binary of 0 and 255 values.

I used the ImageJ’s libraries so as to get used to using it. I used ImageJ libraries to get the image from a directory and place it into a 2d int array where I conduct a simple form of thresholding on the greyscale value of 127 to get a binary image which is my binaryThresholding method.



Since I decided to use white as the background, mostly because colored components against white seemed easier to look at for me, some pictures needed to be inversed to make the components black. In the above code I just flipped the greaterthan to a lesserthan when that happened.

That would result in something like this:

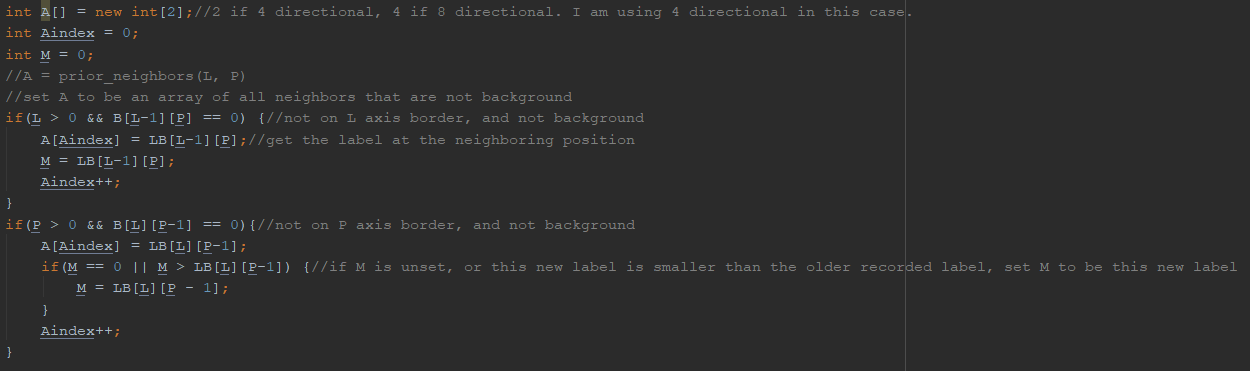


There’s noise, but it works fine for finding connected components.

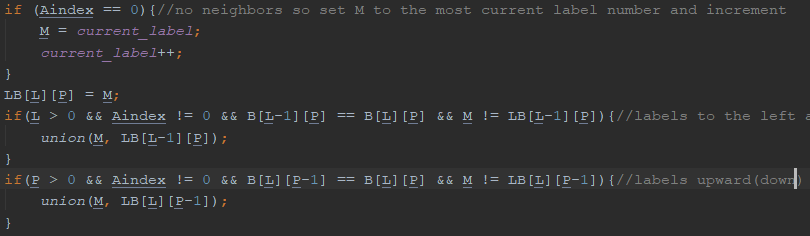
For finding connected components I followed the pseudocode from the Shapiro and Stockman paper, however instead of implamenting the union-find data structure I represent it with an int array named parents and move around some of the code to fit it. Since I already had the pixels in a double array it seemed overly tedious to make a data structure for this implementation seeing as with arrays it still compiles nearly instantly.

I kept the initialize method the same, however with my array implementation I could have done without it all together.

My classical\_with\_union\_find method follows the same variable names as the pseudocode for the sake of clarity. For the A := prior\_neighbors(L, P); part I iterate through my labeled image and using 4 directional searching I look at the labels at the left and bellow the current pixel. This is because when the image is placed by ImageJ into the double array the image is rotated so that the top left of the image is at B[0][0]. So in reality it is looking at the pixel above and to the left.



Then I use union on the labels to the left and down if they are different to the current pixel’s label.

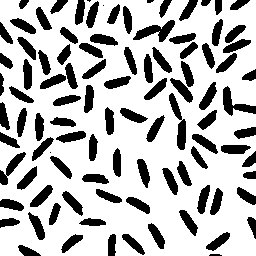
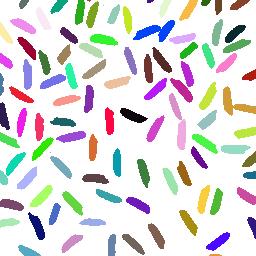


The second pass is the same as the pseudocode.

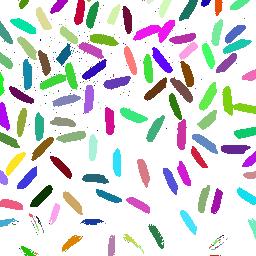
After the iterations are done it returns the double array of labels. It then sets each unique label to the same color and the background stays white inside of a array of size width\*height named pixels. The colors are all random, however I make sure to keep a list of used colors and the label they represent in a hashmap so that they are all unique. With how many possible colors there are I have not had any problems with components seeming too similar. The program prints the number of connected components for clarity. The pixel array is then put into a buffered image and is written to file.

**Results:**

**Binary image:**



**Greyscale image:**

****