CSC 365 Assignment 2

Image Processing with Hashing

Student: Jimmy Nguyen **T,TH** 12:45 PM – 2:05 PM

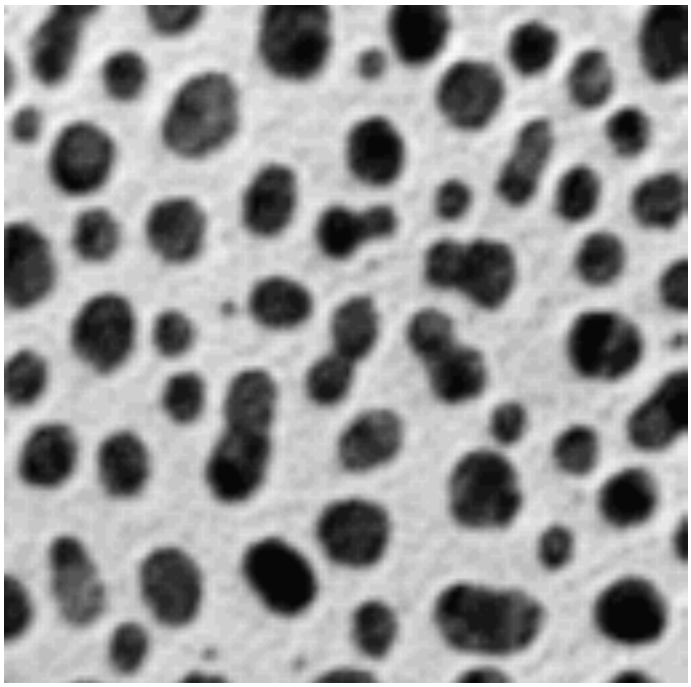
Table of Contents

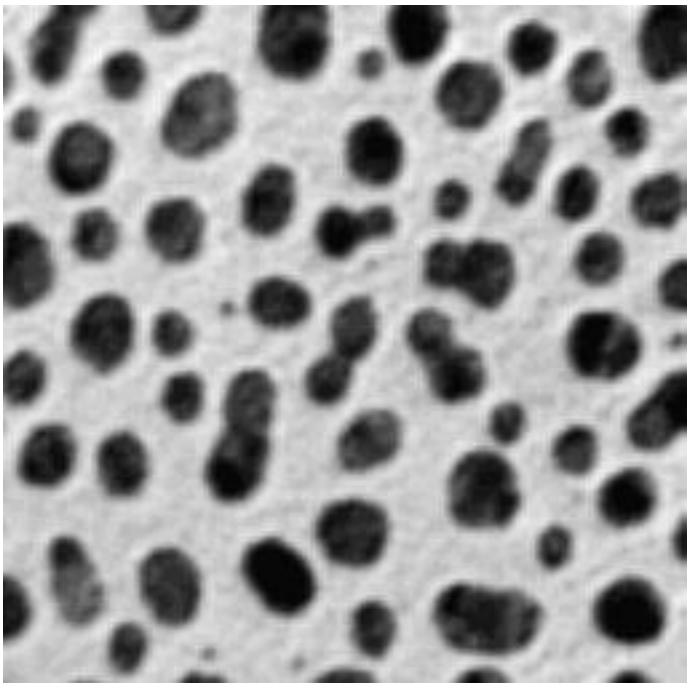
Examples are arranged original (top left), greyscale (top right), bw (bottom left), and ccl (bottom right)

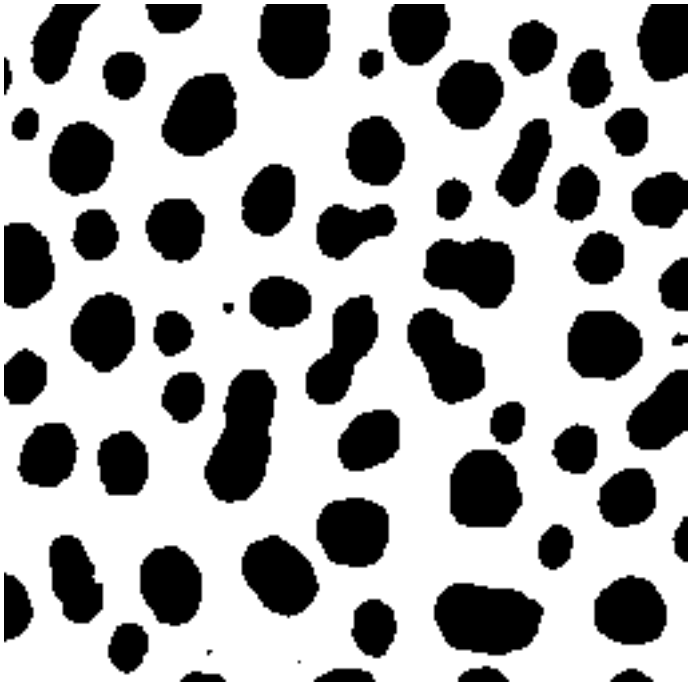
Code

References:

a. https://memorynotfound.com/convert-image-black-white-java/
b. https://alvinalexander.com/blog/post/java/getting-rgb-values-for-each-pixel-in-image-using-java-bufferedi
c. https://processing.org/tutorials/pixels/
d. https://www.geeksforgeeks.org/hashset-in-java/

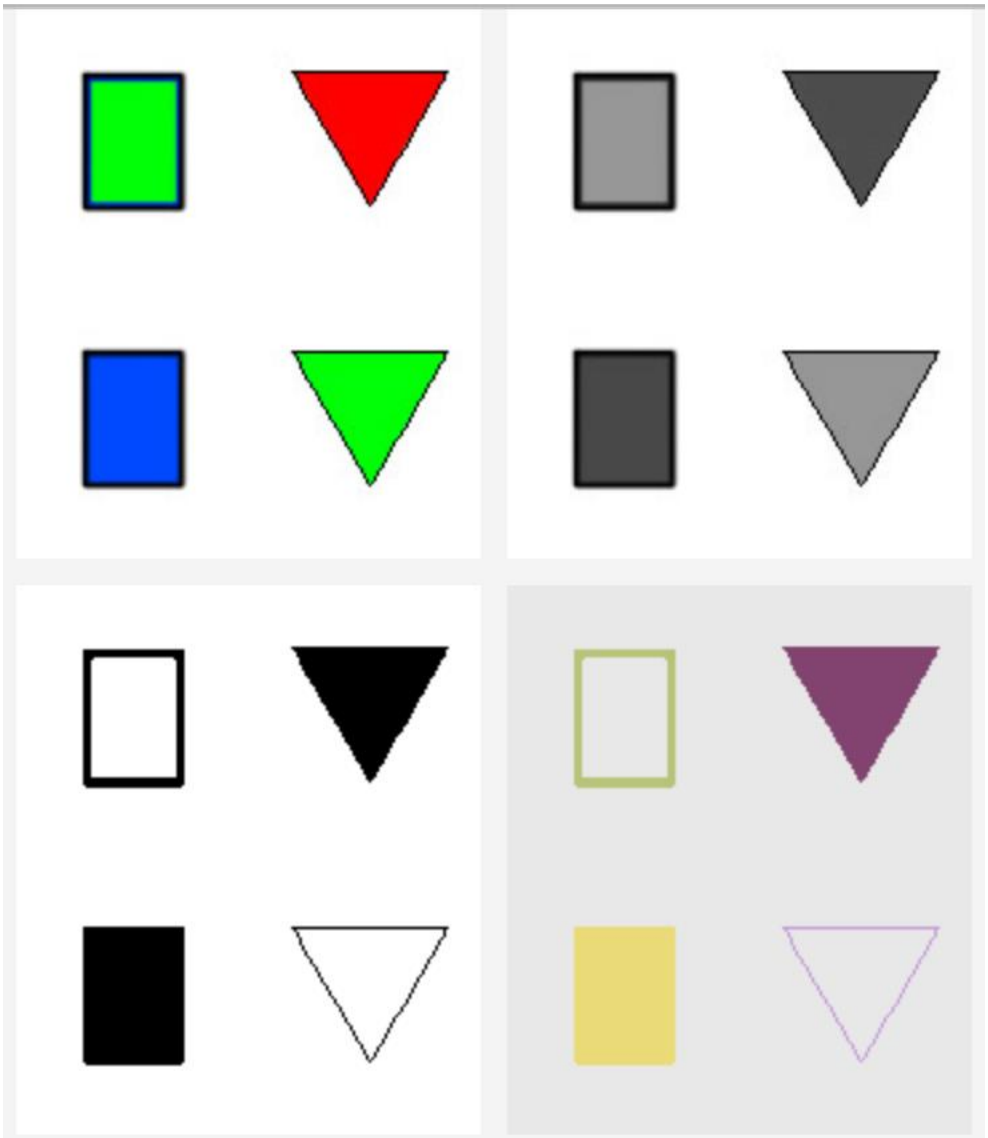






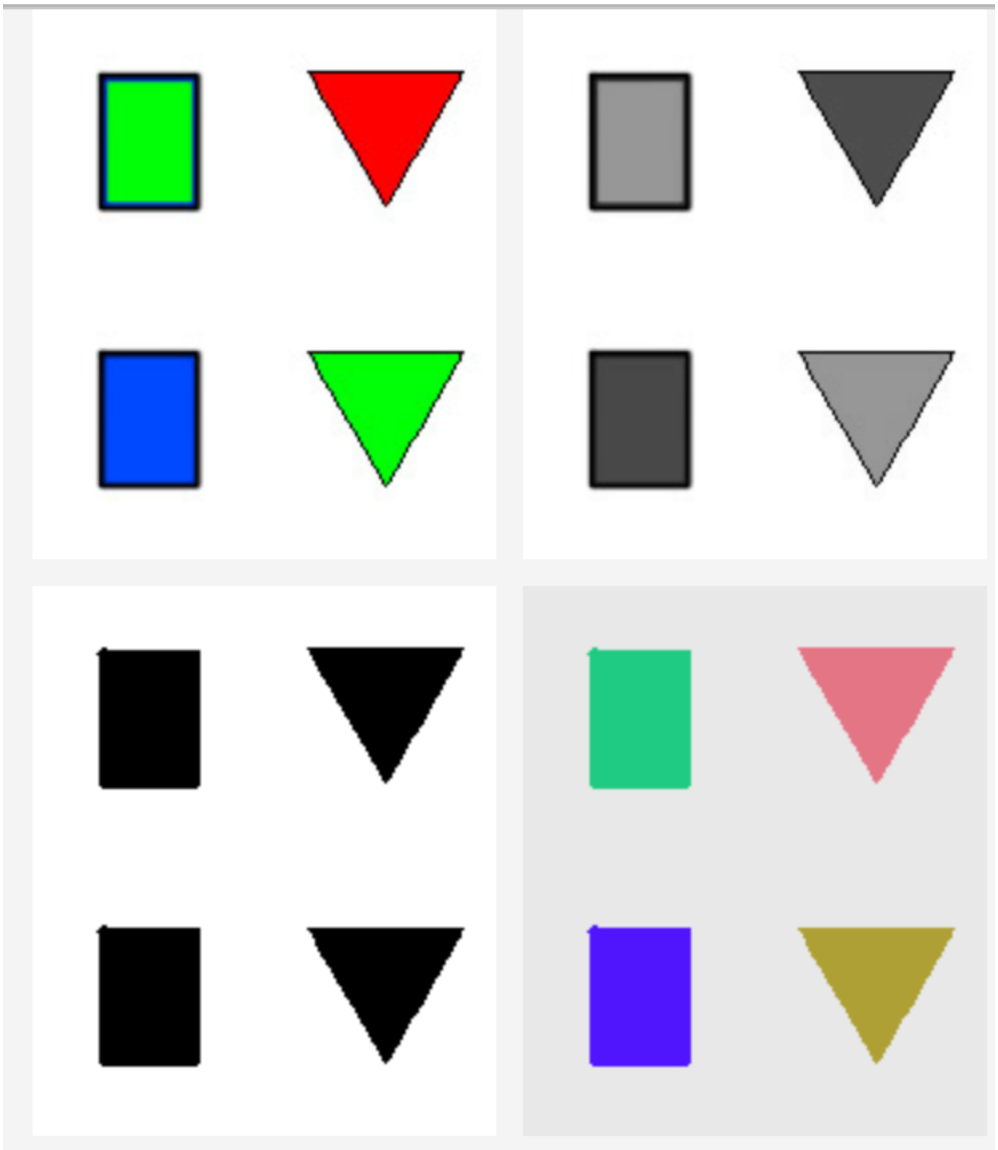
Blobs

Original,
Greyscale,
Black and White,
Connected Component Labeling



Shapes

Original,
Greyscale,
Black and White,
Connected Component Labeling

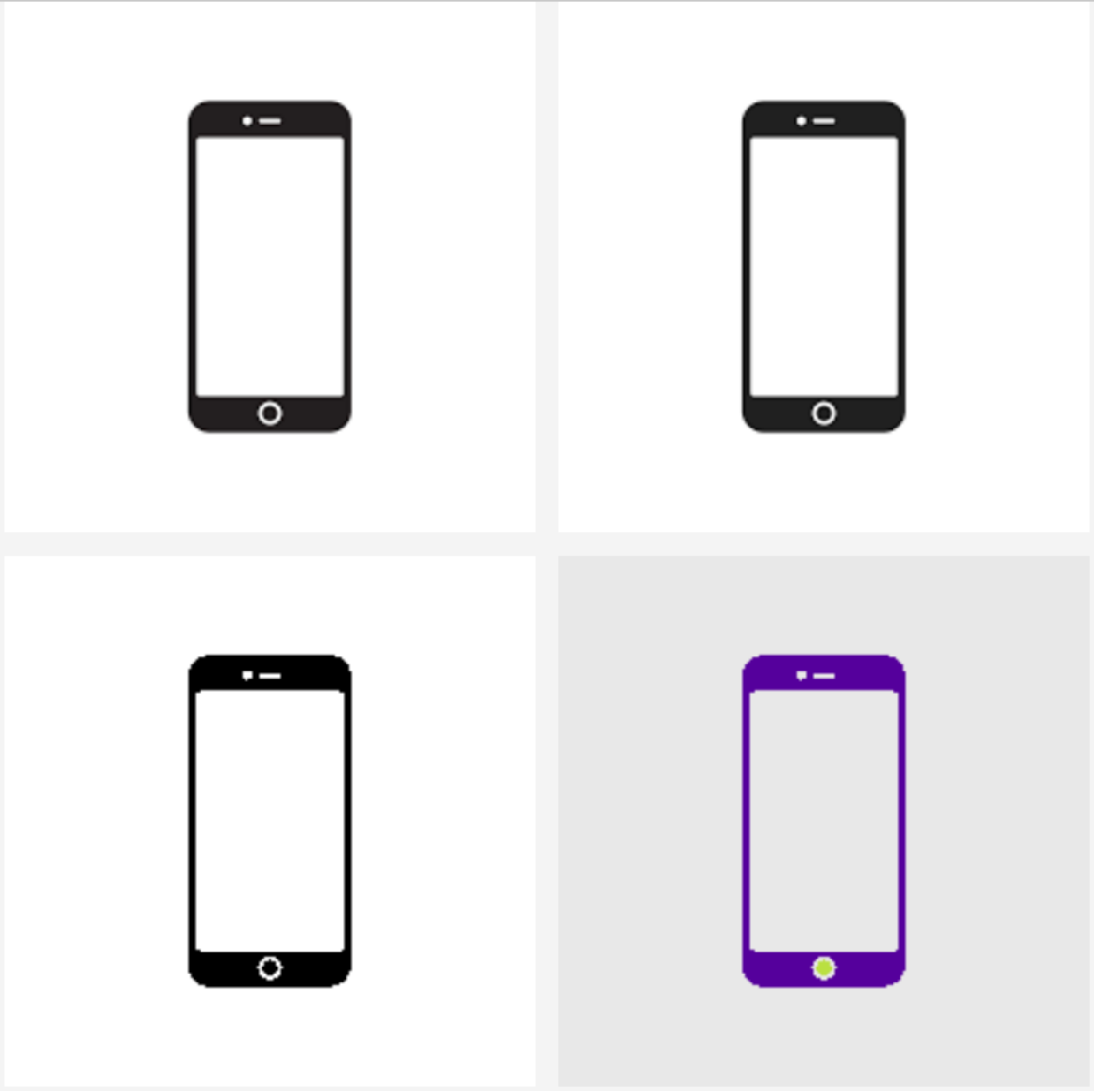


Shapes with Otsu Thresholding

Original,
Greyscale,
Black and White,
Connected Component Labeling

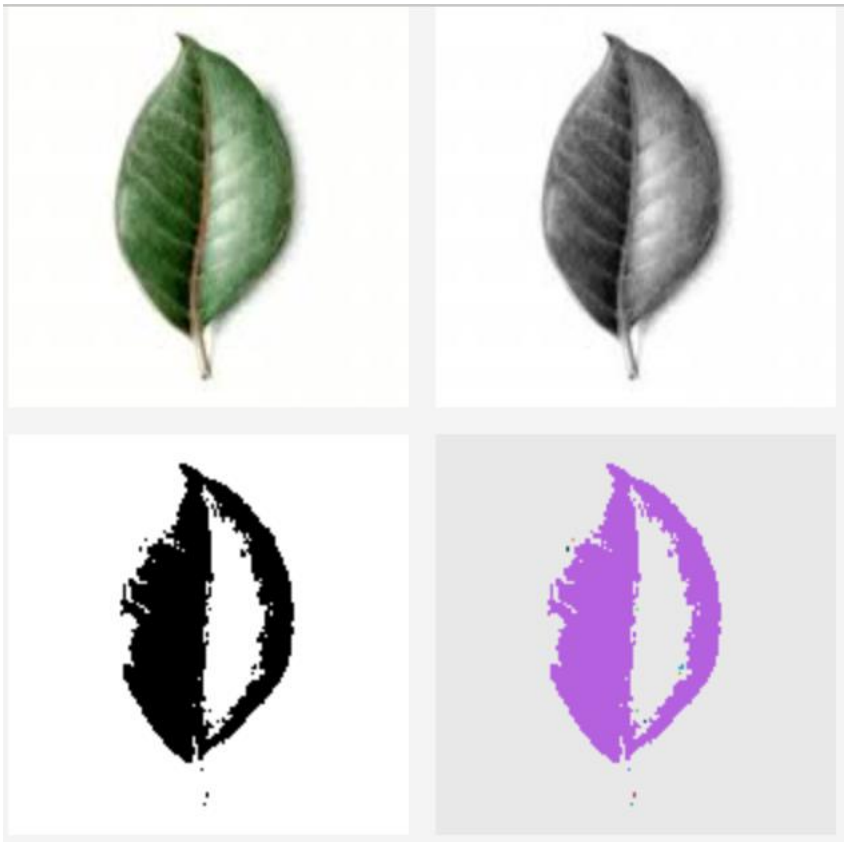
ZULOGO

Original,
Greyscale,
Black and White,
Connected Component Labeling



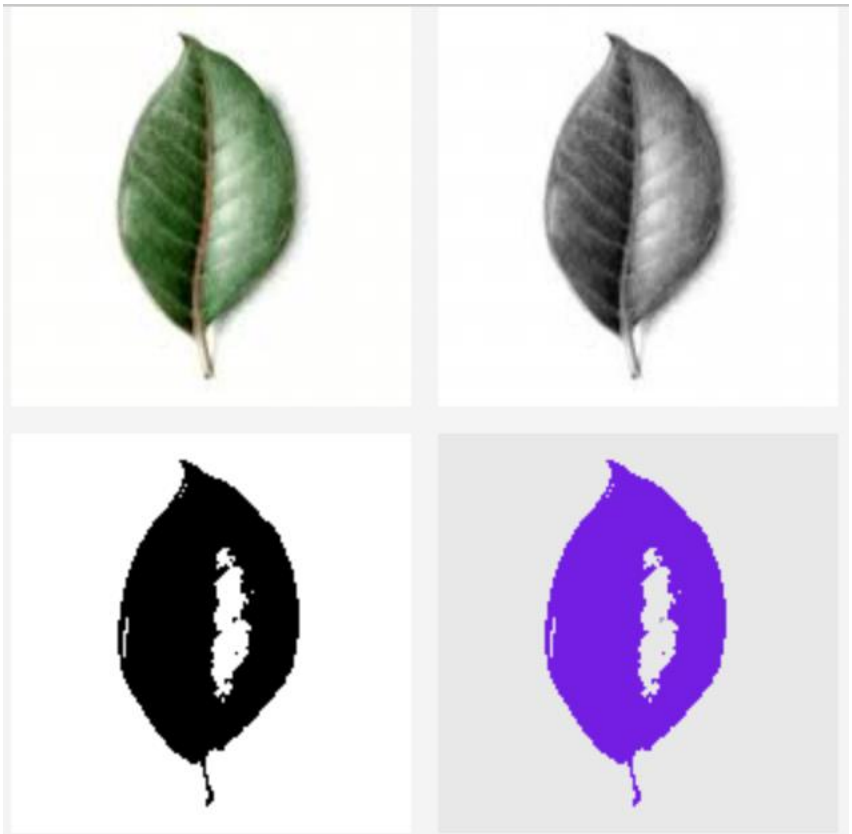
Phone

Original,
Greyscale,
Black and White,
Connected Component Labeling



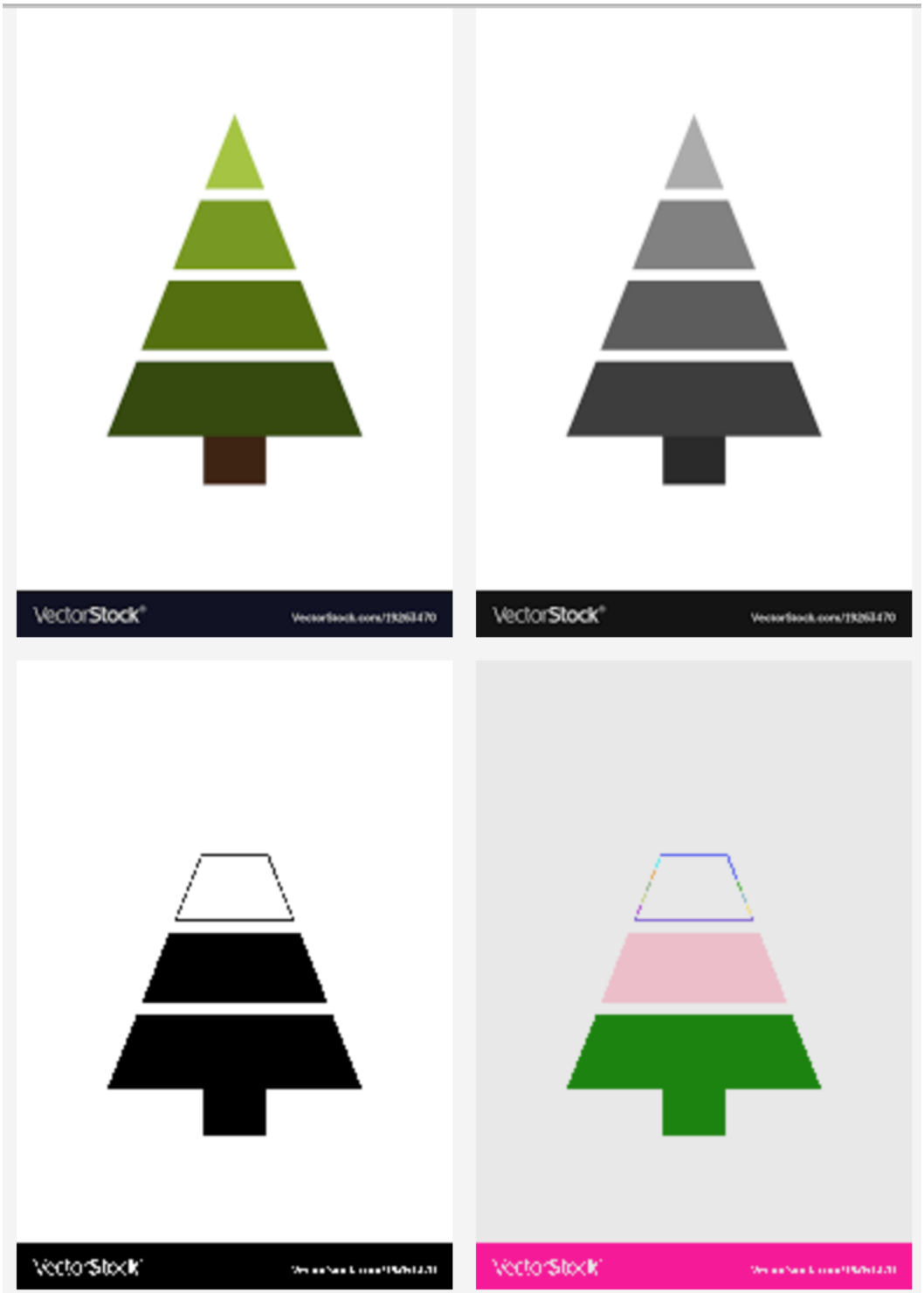
Leaf

Original,
Greyscale,
Black and White,
Connected Component Labeling



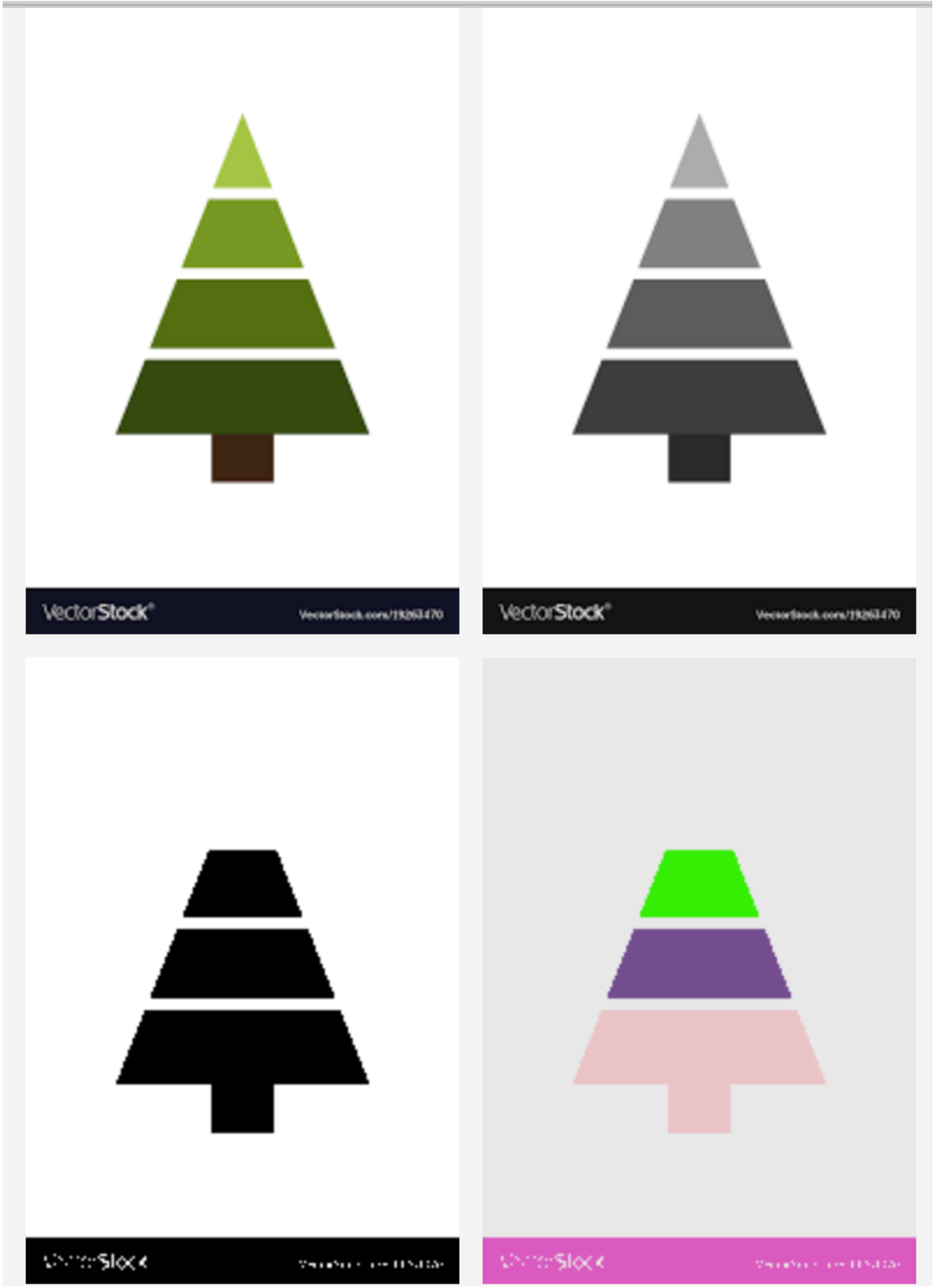
Leaf with Otsu Thresholding

Original,
Greyscale,
Black and White,
Connected Component Labeling



Tree

Original, Greyscale, Black and White, Connected Component Labeling



Tree with Otsu Thresholding

Original, Greyscale, Black and White, Connected Component Labeling

Image Dimensions with Otsu Thresholding

Original,
Greyscale,
Black and White,
Connected Component Labeling

```
public class Driver extends Application {

	// change file paths into /binary, /ccl, /greyscale

	public static void main(String[] args) {
		launch(args);
	}

	@Override
	public void start(Stage primaryStage) throws Exception {
		Scanner input = new Scanner(System.in);


		System.out.println("\t[ Jimmy's CCL ]\n-----------------------------------------------");
		System.out.print("Clear prior files? [y/n]\n>: ");
		if (input.nextLine().toLowerCase().equals("y")) {
			eraseFiles(new File("images/greyscale"));
			eraseFiles(new File("images/binary"));
			eraseFiles(new File("images/ccl"));
			eraseFiles(new File("analysis/"));
		}
		String answer = "";

		while (!answer.equals("exit")) {
			JIP.lc.label = 1;
			System.out.print("Enter an image file: ");
			String fileSrc = input.nextLine();
			System.out.println("-----------------------------------------------\n\n");

			File img = new File(fileSrc);
			if (img.isFile()) {
				File imgGrey = JIP.greyThresholding(img);
				File imgBin = JIP.binarizedImage(imgGrey);
//				File imgBin = Otsu.threshold(imgGrey.toString());
				int[][] binaryAnalysis = JIP.traversePixels(ImageIO.read(imgBin));
				binaryAnalysis = JIP.calculateCCL(binaryAnalysis);
				binaryAnalysis = JIP.ps(binaryAnalysis);
				JIP.save(binaryAnalysis);


				//binaryAnalysis = JIP.passThrough(binaryAnalysis);
				// this won't display until end because stupid JFX thread different so don't make obj.

				//arr list, add it. index = lowest num.

				//binaryAnalysis = JIP.ps(binaryAnalysis);




//				for (String item : ls) {
//					String[] line = item.split(",");
//					if (Integer.valueOf(line[2]) != 0 && binaryAnalysis[])
//						val = Integer.valueOf(line[2]);
//
//					System.out.print(item + "\t");
//				}


				new DisplayWindow(primaryStage, binaryAnalysis, img, imgGrey, imgBin);
			} else {
```

```
                                        System.out.println("Invalid file path entered.");
                                }

                                System.out.println("\n\nContinue or [exit]?");
                                answer = input.nextLine();

                        }
                        input.close();
                }


                private static void eraseFiles(File dir) {
                        for (File file : dir.listFiles()) {
                                if (file.isDirectory())
                                        eraseFiles(file);
                                file.delete();
                        }
                }

}
```

```
public class DisplayWindow {

        public DisplayWindow(Stage x, int[][] binaryAnalysis, File img, File imgGrey, File imgBin) {
                Stage stage = new Stage();
                ScrollPane scroll = new ScrollPane();
                StackPane sp = new StackPane();
                VBox paneP = new VBox(10);

                HBox paneA = new HBox(10);
                HBox paneB = new HBox(10);
                Image image = new Image("file:" + img.toString());
                Image greyImage = new Image("file:images/greyscale/" + imgGrey.getName());
                Image binaryImage = new Image("file:images/binary/" + imgBin.getName());
                Image cclImage = cclWindow(image, binaryAnalysis, img);
```

```
		paneA.getChildren().addAll(new Text(""), new ImageView(image), new ImageView(greyImage), new Text(""));
		paneB.getChildren().addAll(new Text(""), new ImageView(binaryImage), new ImageView(cclImage), new Text(""));
		paneP.getChildren().addAll(paneA, paneB);

		paneA.setAlignment(Pos.CENTER);
		paneB.setAlignment(Pos.CENTER);
		paneP.setAlignment(Pos.CENTER);

		sp.getChildren().add(paneP);


		Label oLb = new Label("Original");
		Label gLb = new Label("Greyscale");
		Label bLb = new Label("Binary");
		Label cLb = new Label("CCL");
		sp.getChildren().addAll(oLb, gLb, bLb, cLb);

		StackPane.setAlignment(oLb, Pos.TOP_LEFT);
		StackPane.setAlignment(gLb, Pos.TOP_RIGHT);
		StackPane.setAlignment(bLb, Pos.BOTTOM_LEFT);
		StackPane.setAlignment(cLb, Pos.BOTTOM_RIGHT);
		StackPane.setAlignment(paneP, Pos.CENTER);
		scroll.setContent(sp);

		sp.minWidthProperty().bind(
				Bindings.createDoubleBinding(() -> scroll.getViewportBounds().getWidth(),
scroll.viewportBoundsProperty()));

		stage.setMinHeight(image.getHeight() * 2);
		stage.setMinWidth(image.getWidth() * 2);
		stage.setScene(new Scene(scroll, 150, 150));
		stage.setTitle("Jimmy's CCL");
		//stage.setResizable(false);
		stage.show();
	}

	public Image cclWindow(Image image, int[][] binaryAnalysis, File img) {
		int width = (int) image.getWidth();
		int height = (int) image.getHeight();

		WritableImage wImage = new WritableImage(width, height);
		PixelWriter writer = wImage.getPixelWriter();

		ArrayList<Color> colorList = new ArrayList<>();


		for (int label = 1; label < JIP.lc.label; label++) {
			// generate a random colour here that doesn't already exist
			Random random = new Random();
			Color randomColor = Color.rgb(random.nextInt(255), random.nextInt(255), random.nextInt(255));

			// make sure not already exist or make another
			for (int i = 0; i < colorList.size(); i++)
				while (randomColor == colorList.get(i))
					randomColor = Color.rgb(random.nextInt(255), random.nextInt(255),
random.nextInt(255));

			for (int x = 0; x < binaryAnalysis.length; x++)
				for (int y = 0; y < binaryAnalysis[0].length; y++)
					if (binaryAnalysis[x][y] == label)
						writer.setColor(y, x, randomColor);
		}
```

```
                try {
                        String ext = "";
                        int i = img.getName().lastIndexOf('.');
                        if (i > 0)
                           ext = img.getName().substring(i+1);

                        File output = new File("images/ccl/ccl_" + img.getName());
                        ImageIO.write(SwingFXUtils.fromFXImage(wImage, null), ext, output);

                        System.out.println("Exporting Image....\n"
                        + "-------------------------------------------------\n"
                                        + "[ " + output.getName() + " ] rendered at " + output.length() + " KB.");

                } catch (IOException e) {
                        e.printStackTrace();
                }

//                 fill in white space. colour is always light grey.
                PixelReader reader = wImage.getPixelReader();
                for (int x = 0; x < width; x++)
                        for (int y = 0; y < height; y++)
                                if (reader.getColor(x, y).toString().equals("0x00000000"))
                                        writer.setColor(x, y, Color.rgb(232, 232, 232)); //
        System.out.println(color.toString());

                return wImage;
        }

}

public class LabelCounter {

        public int label;

        public LabelCounter() {
                label = 1;
        }

}
```

```
public class JIP {

	public static LabelCounter lc = new LabelCounter(); // value vs ref

	public static File greyThresholding(File img) {
		File output = null;
		BufferedImage image;
		try {
			image = ImageIO.read(img);
			BufferedImage gsImage = new BufferedImage(image.getWidth(), image.getHeight(),
BufferedImage.TYPE_BYTE_GRAY);
			Graphics2D g = gsImage.createGraphics();
			g.drawImage(image, 0, 0, null);
			g.dispose();

			output = new File("images/greyscale/gs_" + img.getName());
	ImageIO.write(gsImage, "png", output);
	System.out.println("GS Image processing\n"
		+ "-------------------------------------------------\n"
		+ "[ " + output.getName() + " ] rendered at " + output.length() + " KB.\n\n");
		} catch (IOException e) {
			e.printStackTrace();
		}

		return output;
	}

	public static File binarizedImage(File img) {
		try {
	BufferedImage image = ImageIO.read(img);
```

```
        System.out.println("File Information:"
                        + "\n--------------------------------------------------"
                                        + "\nFile Name: " + img.getName()
                                        + "\nFile Size: " + img.length() + " KB"
                                        + "\nPixel Size: " + image.getWidth() + " x " + image.getHeight()
                                        + "\n--------------------------------------------------\n\n");

        //BufferedImage binaryImg = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.TYPE_BYTE_GRAY); GREY
SCALE
        BufferedImage binaryImg = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.TYPE_BYTE_BINARY);

        Graphics2D graphic = binaryImg.createGraphics();
        graphic.drawImage(image, 0, 0, Color.WHITE, null);
        graphic.dispose();

        File output = new File("images/binary/bin_" + img.getName());
        ImageIO.write(binaryImg, "png", output);                                        // get extension here
        System.out.println("Bin Image processing\n"
                        + "-------------------------------------------------\n"
                        + "[ " + output.getName() + " ] rendered at " + output.length() + " KB.\n\n");
        return output;
    } catch (IOException e) {
        e.printStackTrace();
    }

                        System.out.println("Something went wrong...\n\n");
                        return null;
            }

            public static int[][] traversePixels(BufferedImage image) {
                        System.out.println("Connected Component Labeling....\n"
                        + "-------------------------------------------------\n"
                        + "Analyzing pixel data...\n");

                        int[][] binaryAnalysis = new int[image.getHeight()][image.getWidth()];

                        for (int y = 0; y < image.getHeight(); y++) {
                                    for (int x = 0; x < image.getWidth(); x++) {
                                int pixel = image.getRGB(x, y);

                                    if (((pixel >> 16) & 0xff) == 0) // red pixel // FB reversing this works?
                                                binaryAnalysis[y][x] = 1;
                                    else
                                                binaryAnalysis[y][x] = 0;
                                    }
                        }

//                      System.out.println("Exporting ASCII representation...\n");
//
//                      for (int i = 0; i < binaryAnalysis.length; i++) {
//                                  System.out.print(i + "\t");
//                                  for (int j = 0; j < binaryAnalysis[i].length; j++) {
//                                              if (binaryAnalysis[i][j] == 0) {
//                                                          System.out.print(".");
//                                              } else {
//                                                          System.out.print("B");
//                                              }
//                                  }
//                                  System.out.println("");
//                      }

                        return binaryAnalysis;
```

```
	}

	public static int[][] calculateCCL(int[][] binaryAnalysis) {
		System.out.println("\n\nCalculating CCL\n"
		+ "-------------------------------------------------");


		int[][] arr = hoshenKopelmanAlgorithm(binaryAnalysis);

//		int[][] arr = Con.twoPass(binaryAnalysis);

//		arr = JIP.passThrough(arr);

		System.out.println("Exporting Analysis File\n"
		+ "-------------------------------------------------\n"
					+ "Analysis file being written to disk...");

//		save(arr);

		return arr;
	}

//	algorithm TwoPass(data)
//	  linked = []
//	  labels = structure with dimensions of data, initialized with the value of Background
//
//	  First pass
//
//	  for row in data:
//	     for column in row:
//	        if data[row][column] is not Background
//
//	           neighbors = connected elements with the current element's value
//
//	           if neighbors is empty
//	              linked[NextLabel] = set containing NextLabel
//	              labels[row][column] = NextLabel
//	              NextLabel += 1
//
//	           else
//
//	              Find the smallest label
//
//	              L = neighbors labels
//	              labels[row][column] = min(L)
//	              for label in L
//	                 linked[label] = union(linked[label], L)
//
//	  Second pass
//
//	  for row in data
//	     for column in row
//	        if data[row][column] is not Background
//	           labels[row][column] = find(labels[row][column])
//
//	  return labels


	public static int[][] hoshenKopelmanAlgorithm(int[][] binaryImage) {
		ArrayList<Set<Integer>> cclList = new ArrayList<Set<Integer>>(); // link up labels with pixels.
		int[][] labelArr = firstPass(cclList, binaryImage, lc);
		labelArr = secondPass(cclList, binaryImage, labelArr, lc);
```

```
		labelArr = JIP.passThrough(labelArr);

		System.out.println("+ ( HK Algorithm completed )\n");
		return labelArr;
	}

	private static int[][] firstPass(ArrayList<Set<Integer>> cclList, int[][] binaryImage, LabelCounter lc) {
		int[][] labelArr = new int[binaryImage.length][binaryImage[0].length]; // value of bg dimensions

		for (int x = 0; x < binaryImage.length; x++) {

			for (int y = 0; y < binaryImage[0].length; y++) {

				// 0 means white, 1 means black in binary imaging
				if (binaryImage[x][y] != 0) { // is white
					int[] neighbors = getNeighbors(labelArr, x, y);

					if (neighbors.length == 0) {			// if no neighbors
						cclList.add(new HashSet<>());  // saw someone use hashset. You can use it in array list.
						cclList.get(lc.label - 1).add(lc.label); // get first index and set to 1
						labelArr[x][y] = lc.label++; // we will get next label now
					} else {
						Arrays.sort(neighbors);					// sort to asc
						labelArr[x][y] = neighbors[0];		// set to first index of neighbor if not white
						for (int i = 0; i < neighbors.length; i++)
							for (int j = 0; j < neighbors.length; j++)
								cclList.get(neighbors[i] - 1).add(neighbors[j]);
						// find smallest label
					}
				}
			}
		}

		System.out.println("+ ( First pass completed )");
		return labelArr;
	}

	private static int[][] secondPass(ArrayList<Set<Integer>> cclList, int[][] binaryImage, int[][] labelArr, LabelCounter lc) {
		int[] binaryArr = new int[lc.label];

		for (int i = 0; i < lc.label - 1; i++)
			binaryArr[i] = Collections.min(cclList.get(i), null); // smallest elm to take null
		// https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html

		for (int x = 0; x < binaryImage.length; x++)			// if not in background
			for (int y = 0; y < binaryImage[0].length; y++)		// fetch black 1
				if (binaryImage[x][y] != 0)
					labelArr[x][y] = binaryArr[labelArr[x][y] - 1];

		System.out.println("+ ( Second pass completed )");
		return labelArr;
	}

	private static int[] addNeighbor(int[] neighbors, int binaryRepresentation) {
		int[] neighborsCpy = new int[neighbors.length + 1];

		for (int i = 0; i < neighbors.length; i++)
			neighborsCpy[i] = neighbors[i];
```

```
		neighborsCpy[neighborsCpy.length - 1] = binaryRepresentation;

		return neighborsCpy;
	}

	private static int[] getNeighbors(int[][] binaryImage, int x, int y) {
		int[] neighbors = {}; // initialize as null. Will be dynamically adding to it

		if (x == 0 && y == 0) // if in corner so we're looknig down only
			return neighbors;
		else if (y == 0) // if on top
			neighbors = addNeighbor(neighbors, binaryImage[x - 1][y]);
		else if (x == 0) // if on left side
			neighbors = addNeighbor(neighbors, binaryImage[x][y - 1]);
		else if ((y > 0 && x > 0) && (y < binaryImage[0].length - 1)) { // in inner section and not at bottom
			neighbors = addNeighbor(neighbors, binaryImage[x - 1][y + 1]); // has one in upper left corner
			neighbors = addNeighbor(neighbors, binaryImage[x - 1][y]); // has one in left
			neighbors = addNeighbor(neighbors, binaryImage[x - 1][y - 1]); // has one in lower left corner
			neighbors = addNeighbor(neighbors, binaryImage[x][y - 1]); // has one down
		} else if (y > 0 && x > 0) {
	// in inner section
			neighbors = addNeighbor(neighbors, binaryImage[x - 1][y]); // has one in left
			neighbors = addNeighbor(neighbors, binaryImage[x - 1][y - 1]); // has one in lower left corner
			neighbors = addNeighbor(neighbors, binaryImage[x][y - 1]); // one down
		}

		int[] neighborsCpy = {};

		for (int i = 0; i < neighbors.length; i++)
			if (neighbors[i] != 0)
				neighborsCpy = addNeighbor(neighborsCpy, neighbors[i]);
		// taking all the black pixels and return as neighbors

		return neighborsCpy;
	}

	public static int[][] passThrough(int[][] binaryAnalysis) {
		// check the labels. If missing a number then reorder accordingly.

		ArrayList<Integer> lbList = new ArrayList<>();

		for (int x = 0; x < binaryAnalysis.length; x++) {
			for (int y = 0; y < binaryAnalysis[0].length; y++) {
				if (!lbList.contains(binaryAnalysis[x][y])) {
					lbList.add(binaryAnalysis[x][y]);
				}
			}
		}

		lc.label = lbList.size();

		HashMap<Integer, Integer> mapper = new HashMap<>();

		// key = label
		// value = num 0, 1, 2, 3, 4
		for (int i = 1; i < lbList.size(); i++) {
			mapper.put(i, lbList.get(i));
		}

		int counter = 1;
		for (int i = 1; i < lbList.size(); i++) {
```

```
                        System.out.println("KEY: " + i + "\tVALUE: " + mapper.get(i));

                        boolean flagger = false;

                        for (int x = 0; x < binaryAnalysis.length; x++) {
                                for (int y = 0; y < binaryAnalysis[x].length; y++) {
                                        if (binaryAnalysis[x][y] == mapper.get(i)) {
                                                binaryAnalysis[x][y] = counter;
                                                flagger = true;
                                        }
                                }
                        }
                        if (flagger)
                                counter++;
//                      System.out.println(lbList.get(i));
                }

                System.out.println("");


                for (int x = 0; x < binaryAnalysis.length; x++) {
                        for (int y = 0; y < binaryAnalysis[0].length; y++) {
                                boolean flag = binaryAnalysis[x][y] != 0;

                                if (flag && (x != 0 && y != 0)) {

                                        boolean flag2 = binaryAnalysis[x - 1][y] != 0;
                                        boolean flag3 = binaryAnalysis[x][y - 1] != 0;
                                        boolean flag4 = binaryAnalysis[x - 1][y - 1] != 0;

                                        if (binaryAnalysis[x][y] != binaryAnalysis[x - 1][y] && flag2) { // not equal to one
before it

                                                if (binaryAnalysis[x - 1][y] > binaryAnalysis[x][y])
                                                        binaryAnalysis[x - 1][y] = binaryAnalysis[x][y];
                                                else
                                                        binaryAnalysis[x][y] = binaryAnalysis[x - 1][y];
                                        }

                                        if (binaryAnalysis[x][y] != binaryAnalysis[x][y - 1] && flag3) { // not equal to one
before it
                                                if (binaryAnalysis[x][y - 1] > binaryAnalysis[x][y])
                                                        binaryAnalysis[x][y - 1] = binaryAnalysis[x][y];
                                                else
                                                        binaryAnalysis[x][y] = binaryAnalysis[x][y - 1];
                                        }

                                        if (binaryAnalysis[x][y] != binaryAnalysis[x - 1][y - 1] && flag4) { // not equal to one
before it
                                                if (binaryAnalysis[x - 1][y - 1] > binaryAnalysis[x][y])
                                                        binaryAnalysis[x - 1][y - 1] = binaryAnalysis[x][y];
                                                else
                                                        binaryAnalysis[x][y] = binaryAnalysis[x - 1][y - 1];
                                        }
                                }
                        }
                }

//              for (int x = 0; x < binaryAnalysis.length; x++) {
//                      for (int y = 0; y < binaryAnalysis[0].length; y++) {
//                              boolean flag = binaryAnalysis[x][y] != 0;
//
```

```
//                                          if (flag && (x != 0 && y != 0)) {
//
//                                                  boolean flag2 = binaryAnalysis[x - 1][y] != 0;
//                                                  boolean flag3 = binaryAnalysis[x][y - 1] != 0;
//                                                  boolean flag4 = binaryAnalysis[x - 1][y - 1] != 0;
//
//                                                  if (binaryAnalysis[x][y] != binaryAnalysis[x - 1][y] && flag2) { // not equal to one
before it
//                                                          binaryAnalysis[x - 1][y] = binaryAnalysis[x][y];
//                                                  }
//
//                                                  if (binaryAnalysis[x][y] != binaryAnalysis[x][y - 1] && flag3) { // not equal to one
before it
//                                                          binaryAnalysis[x][y - 1] = binaryAnalysis[x][y];
//                                                  }
//
//                                                  if (binaryAnalysis[x][y] != binaryAnalysis[x - 1][y - 1] && flag4) { // not equal to one
before it
//                                                          binaryAnalysis[x - 1][y - 1] = binaryAnalysis[x][y];
//                                                  }
//                                          }
//                                  }
//                          }


                            // this is technically part of pass2
//                          for (int y = 0; y < binaryAnalysis[0].length; y++) {
//                                  for (int x = 0; x < binaryAnalysis.length; x++) {
//                                          boolean flag = binaryAnalysis[x][y] != 0;
//
//                                          if (flag && (x != 0 && y != 0)) {
//                                                  boolean flag2 = binaryAnalysis[x - 1][y] != 0;
//                                                  if (binaryAnalysis[x][y] != binaryAnalysis[x - 1][y] && flag2) {     // not equal to one
before it
//                                                          binaryAnalysis[x - 1][y] = binaryAnalysis[x][y];
//                                                  }
//
//                                                  boolean flag3 = binaryAnalysis[x][y - 1] != 0;
//                                                  if (binaryAnalysis[x][y] != binaryAnalysis[x][y - 1] && flag3) {     // not equal to one
before it
//                                                          binaryAnalysis[x][y - 1] = binaryAnalysis[x][y];
//                                                  }
//
//                                                  boolean flag4 = binaryAnalysis[x - 1][y - 1] != 0;
//                                                  if (binaryAnalysis[x][y] != binaryAnalysis[x - 1][y - 1] && flag4) {                 // not
equal to one before it
//                                                          binaryAnalysis[x - 1][y - 1] = binaryAnalysis[x][y];
//                                                  }
//                                          }
//                                  }
//                          }
//

//                          for (int x = binaryAnalysis.length - 1; x > 0; x--) {
//                                  for (int y = binaryAnalysis[x].length - 1; y > 0; y--) {
//                                          boolean flag = binaryAnalysis[x][y] != 0;
//
//                                          if (flag && (x != binaryAnalysis.length - 1 && y != binaryAnalysis[x].length - 1)) {
//                                                  boolean flag2 = binaryAnalysis[x + 1][y] != 0;
//                                                  if (binaryAnalysis[x][y] != binaryAnalysis[x + 1][y] && flag2) {
//                                                          binaryAnalysis[x + 1][y] = binaryAnalysis[x][y];
//                                                  }
```

```
//
//                                                      boolean flag3 = binaryAnalysis[x][y + 1] != 0;
//                                                      if (binaryAnalysis[x][y] != binaryAnalysis[x][y + 1] && flag3) {
//                                                              binaryAnalysis[x][y + 1] = binaryAnalysis[x][y];
//                                                      }
//
//                                                      boolean flag4 = binaryAnalysis[x + 1][y + 1] != 0;
//                                                      if (binaryAnalysis[x][y] != binaryAnalysis[x + 1][y + 1] && flag4) {             // not
equal to one before it
//                                                              binaryAnalysis[x + 1][y + 1] = binaryAnalysis[x][y];
//                                                      }
//                                              }
//
//                                      }
//                              }


                              // if adjacent aren't the same label value then make them.

//                            for (int y = 0; y < binaryAnalysis[0].length; y++) {
//                                    for (int x = 0; x < binaryAnalysis.length; x++) {          // go through X scan each line then down
//
//                                            boolean flag = binaryAnalysis[x][y] != 0;
//
//                                            // case where in top corner
//
//                                            if (flag && x == 0 && y == 0) {
//                                                    // in top corner....
//                                            } else if (y == 0 & flag) {
//
//                                            } else if (x == 0 & flag) {
//
//                                            } else if (y > 0 && x > 0 && flag && (y < binaryAnalysis[0].length - 1)) {
//
//                                            } else if (y > 0 && x > 0 & flag) {
//                                                    if (binaryAnalysis[x][y] != binaryAnalysis[x - 1][y])
//                                                            binaryAnalysis[x - 1][y] = binaryAnalysis[x][y];
//                                                    else if (binaryAnalysis[x][y] != binaryAnalysis[x + 1][y])
//                                                            binaryAnalysis[x + 1][y] = binaryAnalysis[x][y];
//                                                    else if (binaryAnalysis[x][y] != binaryAnalysis[x][y - 1])
//                                                            binaryAnalysis[x][y - 1] = binaryAnalysis[x][y];
//                                                    else if (binaryAnalysis[x][y] != binaryAnalysis[x][y + 1])
//                                                            binaryAnalysis[x][y + 1] = binaryAnalysis[x][y];
//                                            }
//
//                                            // if not same
////                                          if (binaryAnalysis[x][y] != binaryAnalysis[x][y] && flag) {
////
////                                          }
//
//                                    }
//                            }
//


                              // if adjacent aren't the same label value then make them.
//                            for (int x = 0; x < binaryAnalysis.length; x++) {
//                                    for (int y = 0; y < binaryAnalysis[x].length; y++) {

                                            //boolean flag = binaryAnalysis[x][y] != 0;

//                                          if (x == 0 && y == 0) { // check right and down only
```

```
//                                                      if (binaryAnalysis[x][y] != binaryAnalysis[x + 1][y] && flag)
//                                                              binaryAnalysis[x + 1][y] = binaryAnalysis[x][y];
//                                                      else if (binaryAnalysis[x][y] != binaryAnalysis[x][y + 1] && flag)
//                                                              binaryAnalysis[x][y + 1] = binaryAnalysis[x][y];
//                                              } else if (x == 0) { // check right
//                                                      if (binaryAnalysis[x][y] != binaryAnalysis[x + 1][y] && flag)
//                                                              binaryAnalysis[x + 1][y] = binaryAnalysis[x][y];
//                                              } else if (y == 0) { // check down only
//                                                      if (binaryAnalysis[x][y] != binaryAnalysis[x][y + 1] && flag)
//                                                              binaryAnalysis[x][y + 1] = binaryAnalysis[x][y];
//                                              } else {

//                                                      if (binaryAnalysis[x][y] != binaryAnalysis[x + 1][y] && flag) {
//
//                                                      }
//
//                                                      // case where we have left right up down all dir
//                                                      if (binaryAnalysis[x][y] != binaryAnalysis[x + 1][y] && flag) {
//                                                              binaryAnalysis[x + 1][y] = binaryAnalysis[x][y];
//                                                      } else if (binaryAnalysis[x][y] != binaryAnalysis[x + 1][y] && flag) {
//                                                              binaryAnalysis[x + 1][y] = binaryAnalysis[x][y];
//                                                      } else if (binaryAnalysis[x][y] != binaryAnalysis[x][y + 1] && flag && y !=
binaryAnalysis[x].length - 1) {
//                                                              binaryAnalysis[x][y + 1] = binaryAnalysis[x][y];
//                                                      } else if (binaryAnalysis[x][y] != binaryAnalysis[x][y + 1] && flag && y !=
binaryAnalysis[x].length - 1) {
//                                                              binaryAnalysis[x][y + 1] = binaryAnalysis[x][y];
//                                                      }
//                                              }
//                                      }
//                      }

                        return binaryAnalysis;
        }

        public static int[][] ps(int[][] binaryAnalysis) {
                ArrayList<String> ls = new ArrayList<>();

                for (int x = 0; x < binaryAnalysis.length; x++) {
                        for (int y = 0; y < binaryAnalysis[0].length; y++) {
                                boolean flag = binaryAnalysis[x][y] != 0;

                                if ((x != 0 && y != 0)) {

                                        // while loop? WHILE ADJACENT NOT SAME.
                                        if (x != binaryAnalysis.length - 1 && binaryAnalysis[x + 1][y] != 0 && flag) {
                                                if (binaryAnalysis[x][y] != binaryAnalysis[x + 1][y]) {
                                                        //System.out.println("YUP");
                                                        ls.add(x + "," + y + "," + binaryAnalysis[x + 1][y]);
                                                        binaryAnalysis[x + 1][y] = binaryAnalysis[x][y];
                                                }
                                        }
                                }
                        }
                }

                int val = 1000;

                for (String item : ls) {
                        String[] line = item.split(",");
                        if (Integer.valueOf(line[2]) < val && Integer.valueOf(line[2]) != 0)
                                val = Integer.valueOf(line[2]);
```

```
                        System.out.print(item + "\t");
                }
                System.out.println("\n" + val);


                for (int i = 0; i < ls.size(); i++) {
                        String[] line = ls.get(i).split(",");
                        if (Integer.valueOf(line[2]) != 0) {
                                binaryAnalysis[Integer.valueOf(line[0]) + 1][Integer.valueOf(line[1])] = val;
                        }
                }

                return binaryAnalysis;
        }

        public static void save(int[][] arr) {
                String results = "";


                for (int i = 0; i < arr.length; i++) {
                        for (int j = 0; j < arr[i].length; j++) {
                                results += arr[i][j] + "";
                                //System.out.print(arr[i][j] + " ");
                        }
                        results += "\n";
                        //System.out.println("");
                }

                try {
                        File file = new File("analysis/analysis_" + Instant.now() + ".txt");
                        PrintWriter out = new PrintWriter(file);
                        out.println(results);
                        out.close();
                        System.out.println("\n[ " + file.getName() + " ] rendered at " + file.length() + " KB.\n");
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                }
                        //System.out.println(results);
        }
}
```