CSC 225 - SUMMER 2025
ALGORITHMS AND DATA STRUCTURES I
PROGRAMMING ASSIGNMENT 3
UNIVERSITY OF VICTORIA

Due: Sunday, August 3rd, 2025 before 11:55pm. Submissions will be accepted without penalty until 7:00am on Friday, August 8th, 2025. **No submissions will be accepted after this time.**

1 Overview: Images and Graphs

This assignment covers an application of graph algorithms to image processing. You will implement a data structure to represent images as graphs, with each pixel of the image represented by a vertex. Then, you will apply various graph traversal algorithms to perform various image manipulations, including the classic *flood fill* operation.

Sections 2 and 3 describe a graph abstraction for colour images which uses edges to connect neighbouring pixels which have the same colour. Section 4 summarizes the tasks to perform for the assignment. Section 5 describes the image viewer interface provided with the assignment code.

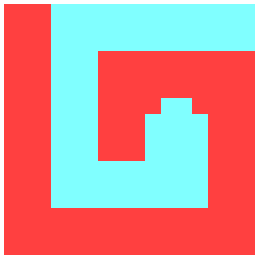
2 Images

Images can be represented by 2-dimensional arrays of pixels, each with a colour value. Usually, colour values are expressed as a triple (r, g, b) of red, green and blue values. Many image processing tasks work entirely with individual pixel values. For example, to convert an image from colour to greyscale, the colour value of each pixel can be converted to a shade of grey. For a pixel with colour (r, g, b), one possible conversion formula is to produce a new colour (r', g', b') where

$$(r', g', b') = \left(\frac{r+g+b}{3}, \frac{r+g+b}{3}, \frac{r+g+b}{3}\right).$$

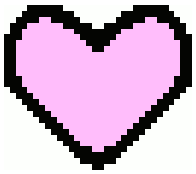
Many image processing tasks use information from neighbouring pixels, or from the entire image. CSC 225 will only cover one technique for image processing.

Consider the small images in Figure 1 below.

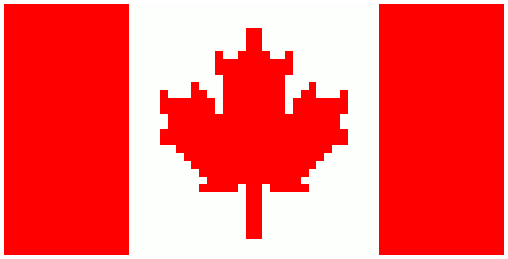


(a) Spiral (16×16)

(b) Rainbow (50×32)



(c) Heart (32×28)



(d) Flag of Canada (64×32)

Figure 1: Four small images.

Each image can be viewed as a grid of pixels. Within an image, groups of adjacent pixels with the same colour are often significant. This assignment covers various algorithms to process contiguous regions inside images. Figures 2, 3 and 4 show the individual pixels of three of the images in Figure 1 above.

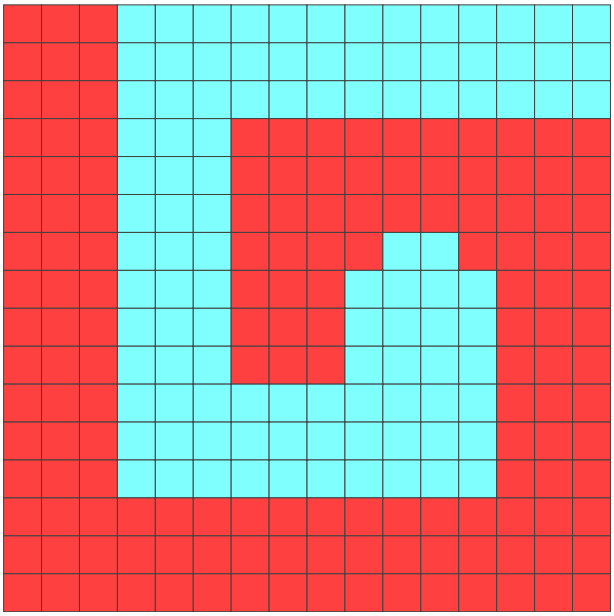


Figure 2: A close-up view of Figure 1a.

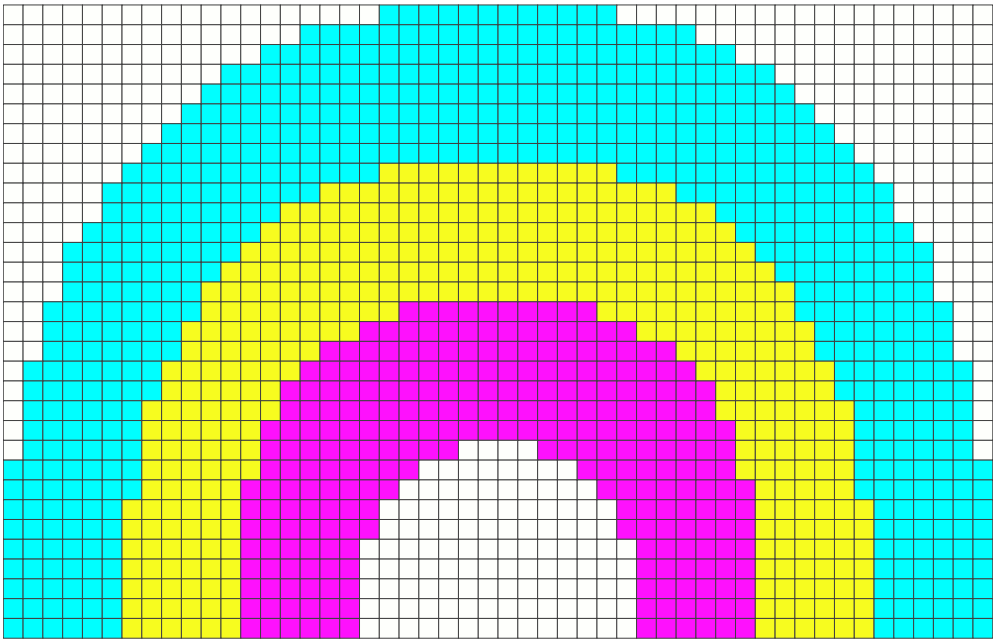


Figure 3: A close-up view of Figure 1b.

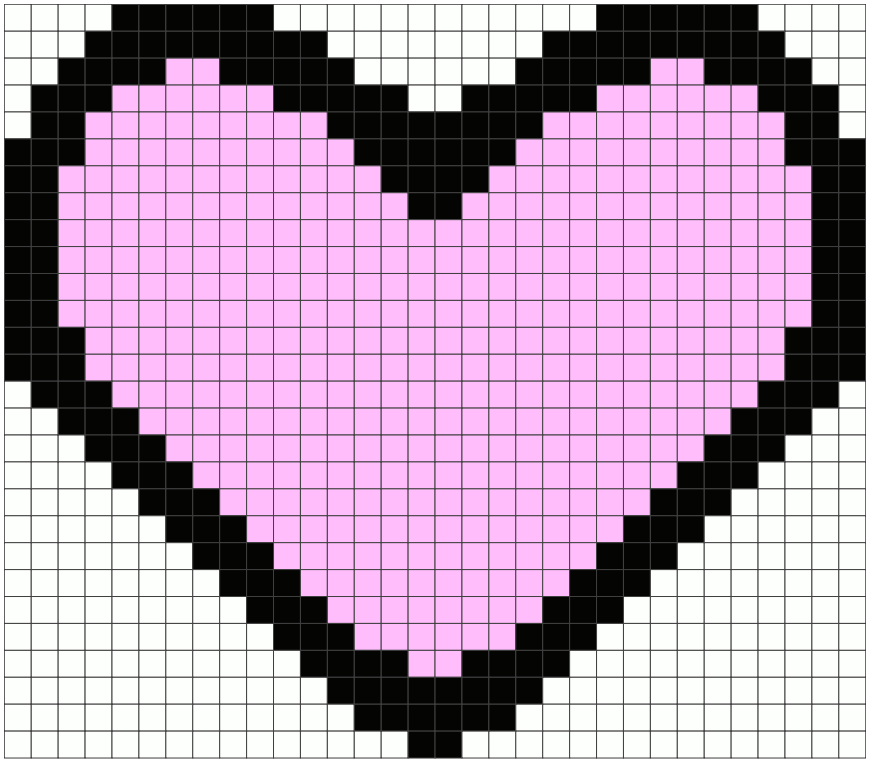


Figure 4: A close-up view of Figure 1c.

3 Pixel Graphs

Define the *pixel graph* of an $n \times m$ pixel image to be an undirected graph with a vertex v_{xy} for each pixel (where $0 \leq x \leq n-1$ and $0 \leq y \leq m-1$), and edges between all pairs of neighbouring pixels with the same colour value. Figures 5, 6 and 7 show the pixel graphs for the images in Figures 2, 3 and 4.

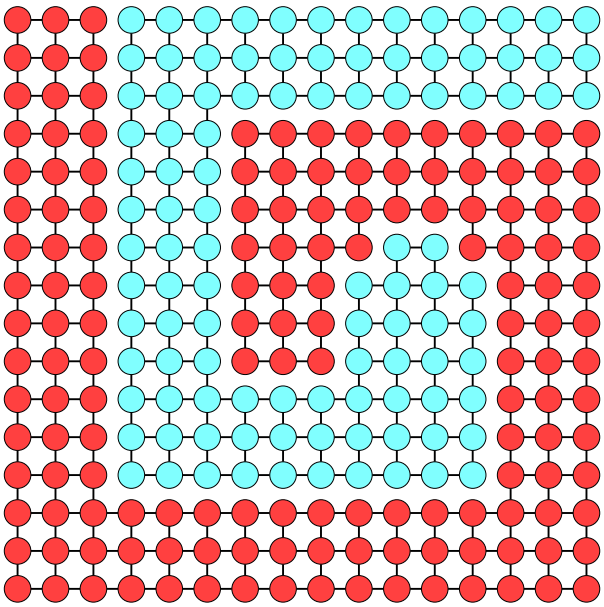


Figure 5: The pixel graph for the image in Figure 1a.

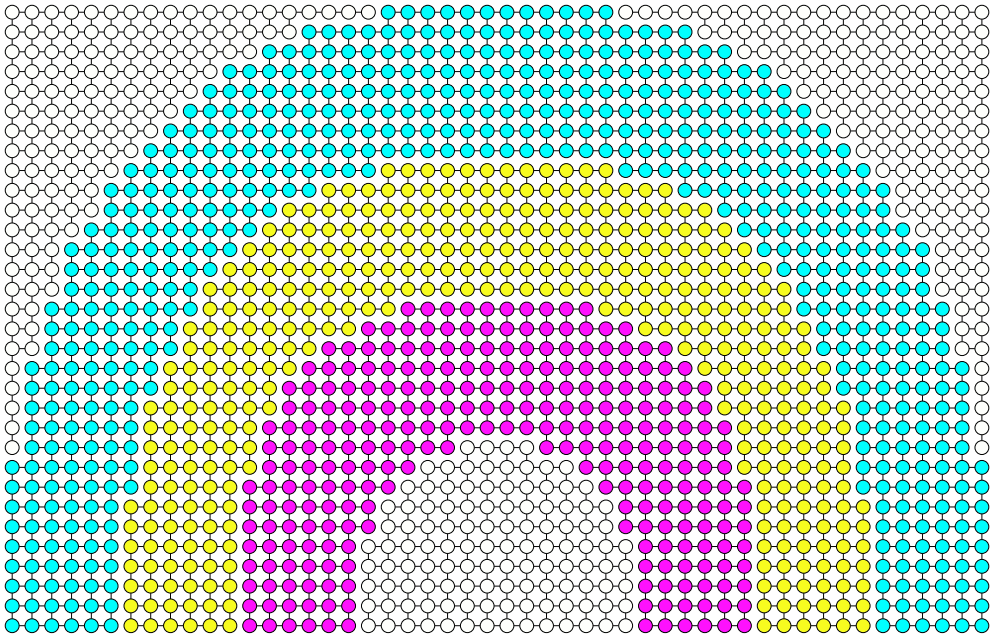


Figure 6: The pixel graph for the image in Figure 1b.

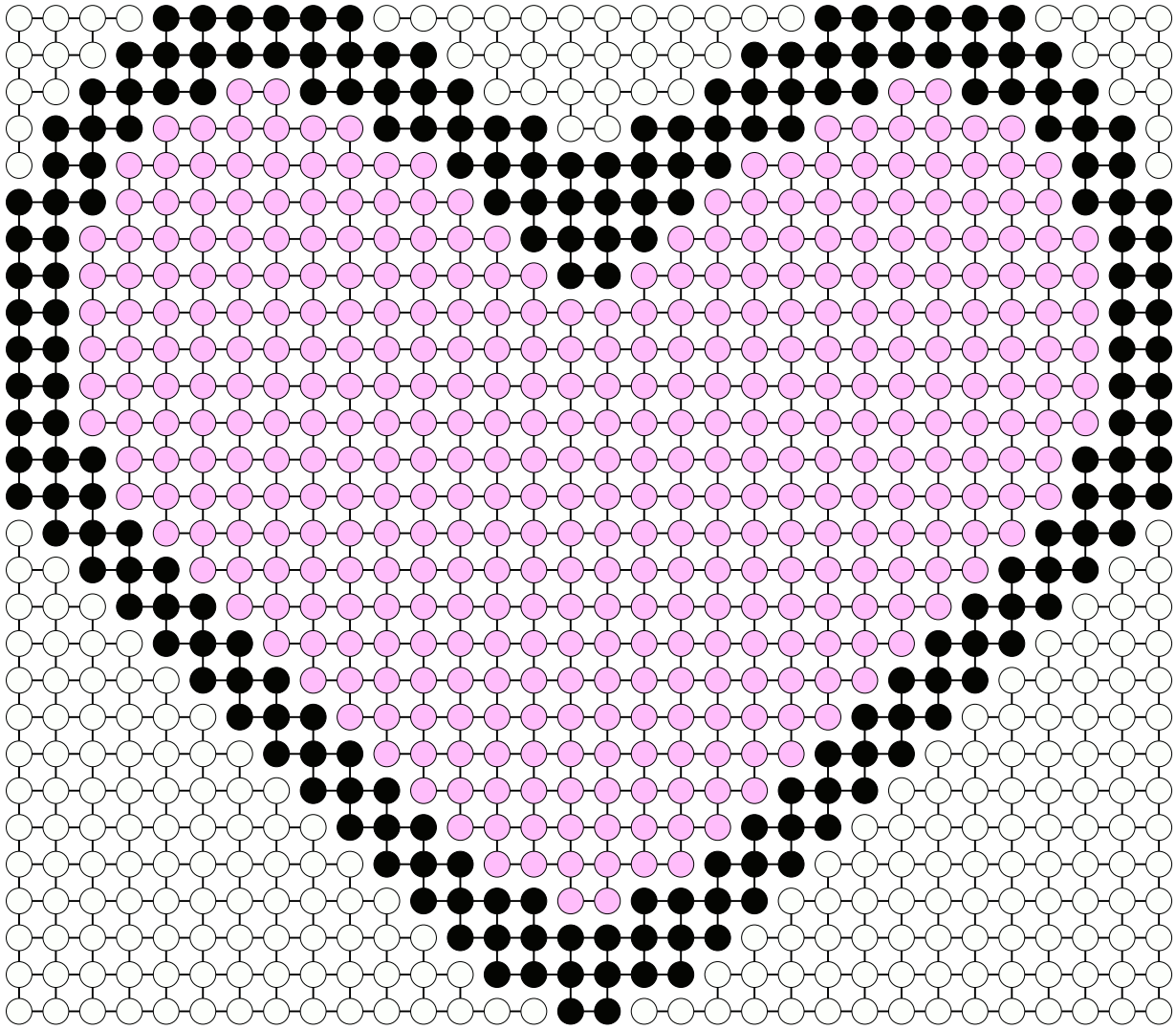


Figure 7: The pixel graph for the image in Figure 1c.

Each connected component of the pixel graph corresponds to a contiguous region containing a single colour in the original image. Vertices with degree 4 correspond to pixels in the interior of their region, while vertices with degree less than 4 correspond to pixels on a boundary.

4 Assignment Structure

Your task for this assignment is to implement a data structure called `PixelGraph` to represent a pixel graph and then implement several graph traversal algorithms using the `PixelGraph` structure. The `PixelGraph` data structure is split between the files `PixelGraph.java` and `PixelVertex.java`. The traversal algorithms will be implemented in static methods in the `A3Algorithms.java` file.

The template for this assignment is divided among five files. You are required to use the provided

files as the basis for your submission, and may not change any of the classnames or method signatures inside any of the files, or your submission will not be marked. You are free to add additional classes, methods or instance variables as needed, except as indicated below. You are also permitted to include extra files in your submission. Note that if your submission is missing any files and does not compile, the error will be treated the same way as any other compile error (and you will not be given the opportunity to submit the missing files to recover the marks).

`A3Algorithms.java`	Contains five static methods: `FloodFillDFS`, `FloodFillBFS`, `OutlineRegionDFS`, `OutlineRegionBFS` and `CountComponents` which perform various tasks on a `PixelGraph` instance. Read the comments in the template for more details. **Methods to implement**: `FloodFillDFS`, `FloodFillBFS`, `OutlineRegionDFS`, `OutlineRegionBFS` and `CountComponents`.
`ImageViewer225.java`	A graphical interface for viewing and transforming images. This is the 'main program' which uses the functionality defined in the other files. It is not necessary to read or understand the code in this file to complete the assignment (although you are encouraged to do so). You are free to modify this program for debugging purposes, but all of your modifications will be discarded before marking. During marking, the original, unmodified `ImageViewer225.java` will be substituted. You will lose marks if your code does not function correctly (or if a compile error occurs) with the unmodified version of `ImageViewer225.java`. **Methods to implement**: None.
`PixelGraph.java`	The `PixelGraph` class implements a data structure for storing a pixel graph. **Methods to implement**: Constructor, `getPixelVertex`, `getWidth`, `getHeight`. (Add other methods or properties as needed)
`PixelVertex.java`	The `PixelVertex` class implements a data structure to store each vertex of a pixel graph. **Methods to implement**: Constructor, `getX`, `getY`, `getNeighbours`, `addNeighbour`, `removeNeighbour`, `getDegree`, `isNeighbour`. (Add other methods or properties as needed)
`PixelWriter.java`	An interface definition used by the methods in `A3Algorithms`. Do not modify this file. **Methods to implement**: None.

5 Image Viewer Interface

After compiling all of the template files, the graphical interface can be started with the command

```
$ java ImageViewer225 image_name.png
```

The Java stack size may need to be modified to accommodate the memory demands of a recursive DFS implementation. In that case, you should use a command like

```
$ java -Xss64m ImageViewer225 image_name.png
```

to increase the stack size (in this case, to 64 megabytes).

Figure 8 shows the image viewer window displaying the file `rainbow.png` (shown in Figure 1b).

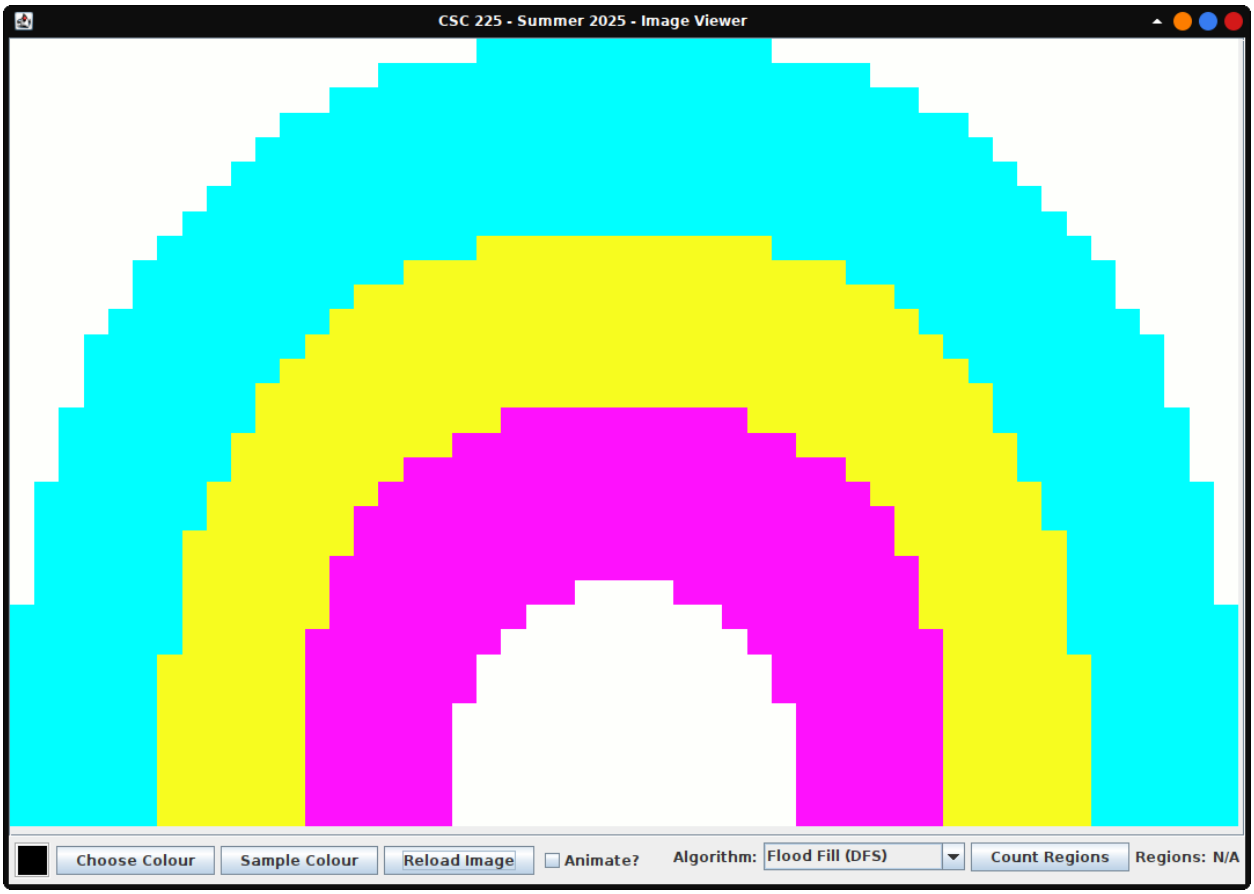


Figure 8: The image viewer window after opening the `rainbow.png` image.

The mouse wheel can be used to control the zoom level of the displayed image (since most of the test images are small, you will likely want to zoom in on them). The "Count Regions" button will construct a PixelGraph object from the displayed image, then use the `CountComponents` function in `A3Algorithms.java` to count the number of connected regions in the image and display the result. Clicking on a pixel of the image will run one of the other algorithms in `A3Algorithms.java`. To select a different colour, use the "Choose Colour" button. The "Sample Colour" button will allow you to set the colour from a pixel in the active image (by left-clicking on the pixel). Figure 9 shows the result of a model solution after clicking at the location shown with the "Flood Fill (DFS)" algorithm selected. Figure 10 shows the result of a model solution after clicking at the location

shown with the "Outline Region (DFS)" algorithm selected. Note that the DFS and BFS versions of both applications (Flood Fill and Outline Region) should produce identical final results.

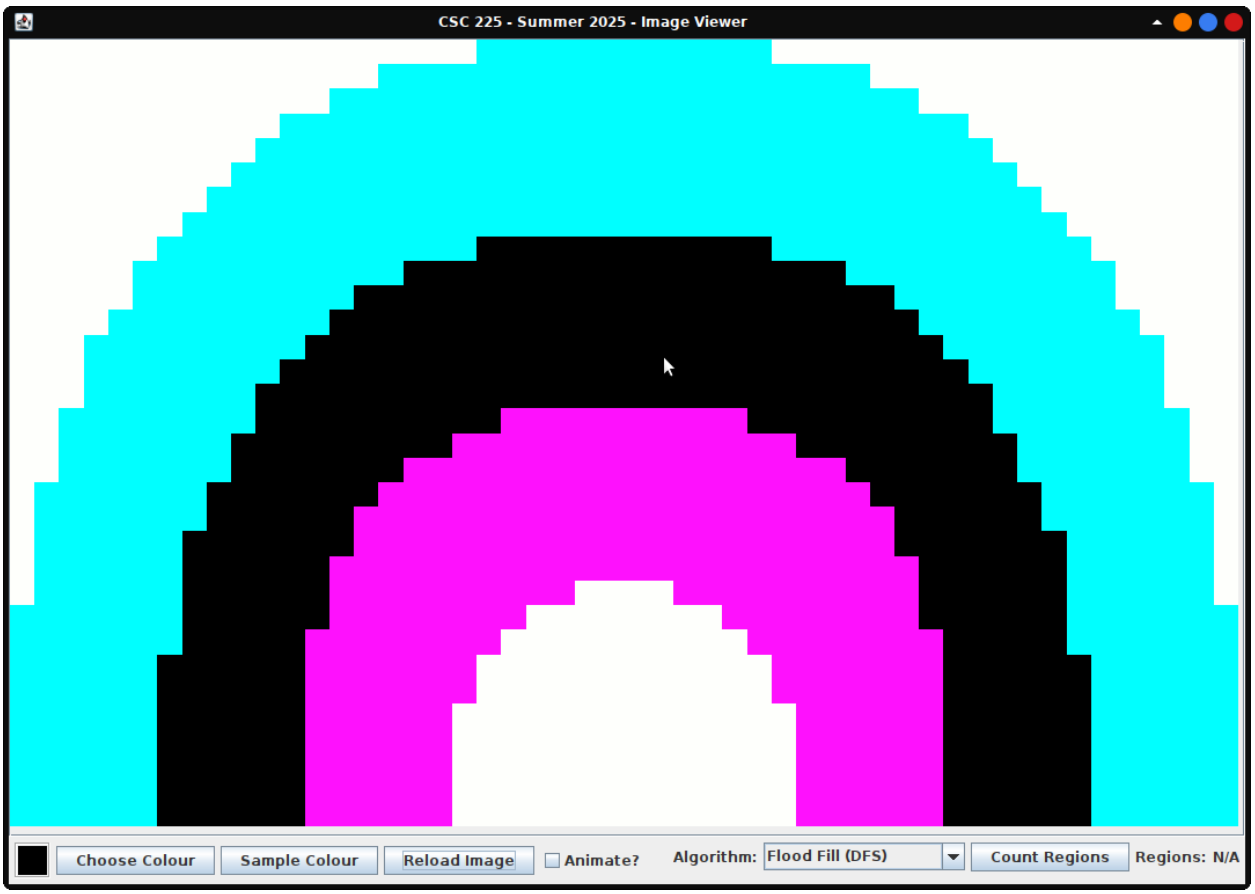


Figure 9: The image viewer window after applying the "Flood Fill (DFS)" algorithm to `rainbow.png` image using black as the fill colour.

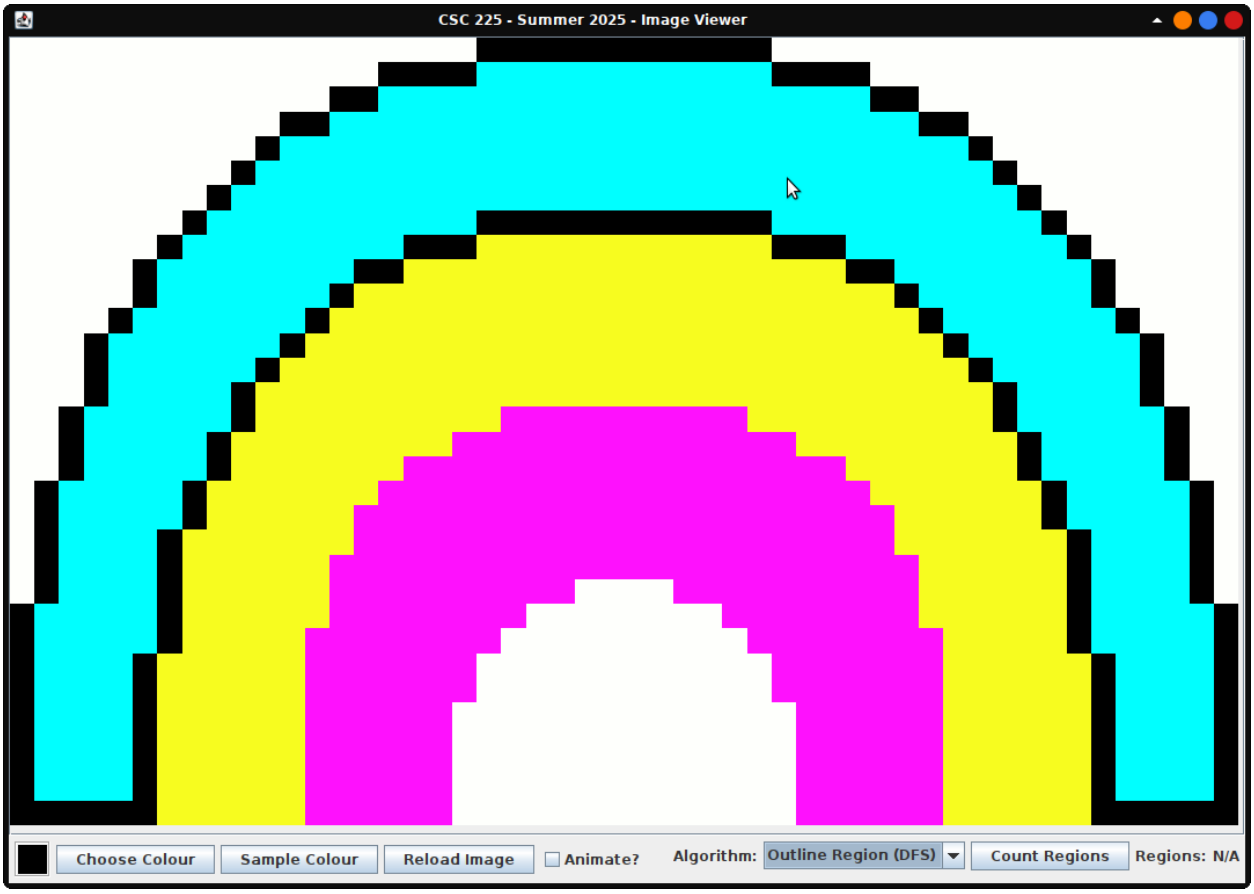


Figure 10: The image viewer window after applying the "Outline Region (DFS)" algorithm to `rainbow.png` image using black as the outline colour.

6 Test Images

A collection of test images (including the images in Figure 1) have been posted. The image viewer is capable of loading any image in PNG format, so you can also use externally-obtained images for testing.

7 Evaluation

Submit all `.java` files needed to compile your assignment electronically via Brightspace. Your code must compile and run correctly in the lab environment (ECS 250). If your code does not compile as submitted, you will receive a mark of zero.

This assignment is worth 5% of your final grade and will be marked out of 12 through a combination of automated testing and human inspection.

The marks are distributed among the components of the assignment as follows.

Marks	Component
2	The `PixelGraph` data structure is implemented correctly.
2	The `FloodFillDFS` method in `A3Algorithms.java` is implemented correctly.
2	The `FloodFillBFS` method in `A3Algorithms.java` is implemented correctly.
1	The `OutlineRegionDFS` method in `A3Algorithms.java` is implemented correctly.
1	The `OutlineRegionBFS` method in `A3Algorithms.java` is implemented correctly.
2	The `CountComponents` method in `A3Algorithms.java` is implemented correctly.
2	In addition to being correct, all of the methods in `A3Algorithms.java` have worst-case O(n) performance on a graph with n vertices.

You are permitted to delete and resubmit your assignment as many times as you want before the due date, but no submissions or resubmissions will be accepted after the due date has passed. You will receive a mark of zero if you have not officially submitted your assignment (and received a confirmation email) before the due date.

At minimum, your submission must include the files `PixelGraph.java`, `PixelVertex.java` and `A3Algorithms.java`. You may use additional files if needed by your solution (as long as the program can be invoked from the command line using the syntax in Section 5). Ensure that each submitted file contains a comment with your name and student number.

Ensure that all code files needed to compile and run your code in ECS 250 are submitted. Only the files that you submit through Brightspace will be marked. The best way to make sure your submission is correct is to download it from Brightspace after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. After submitting your assignment, Brightspace will automatically send you a confirmation email. **If you do not receive such an email, you did not submit the assignment.** If you have problems with the submission process, contact the instructor **before** the due date.