CSC 225 - SUMMER 2015
ALGORITHMS AND DATA STRUCTURES I
PROGRAMMING ASSIGNMENT 5
UNIVERSITY OF VICTORIA

**Due**: Tuesday, July 28th, 2015 at 4:00pm.

1   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. The Department of Computer Science offers an entire course (CSC 205) on 2d graphics and image processing, which covers a broader range of techniques.
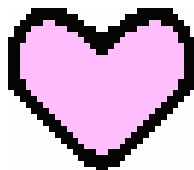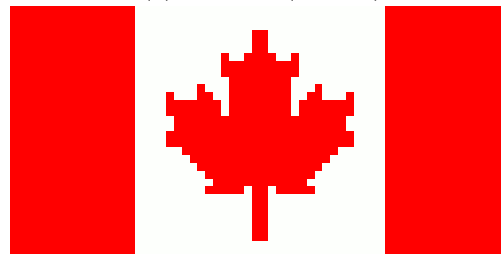
Consider the small images in Figure 1 below.
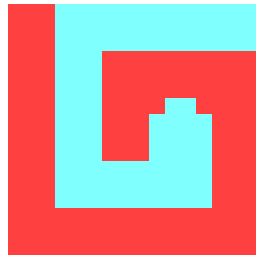
(a) Spiral (16×16)
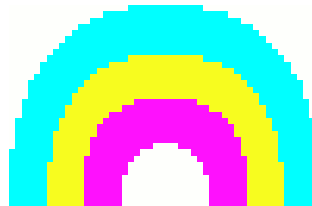
(b) Rainbow (50×32)

(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, and assignment 5, will cover 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.
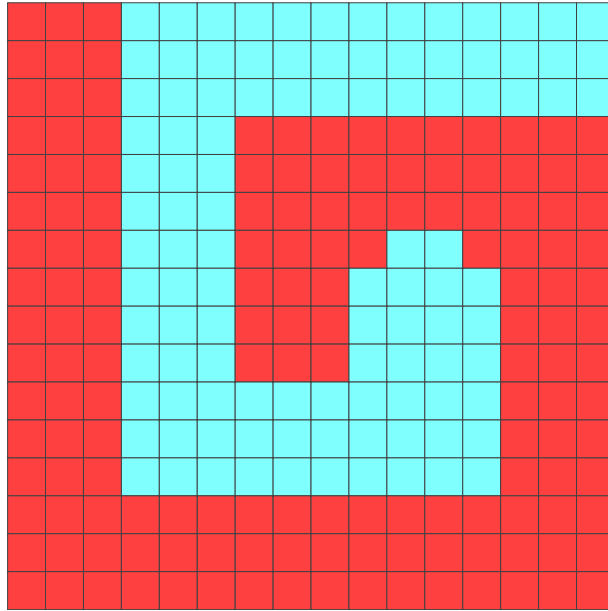

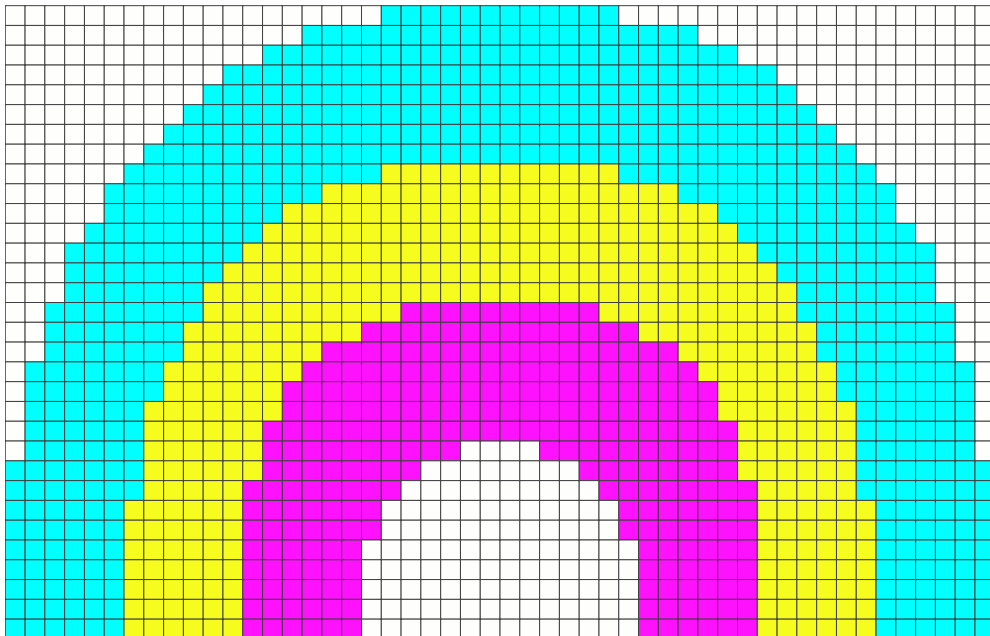Figure 2: A close-up view of Figure 1a.

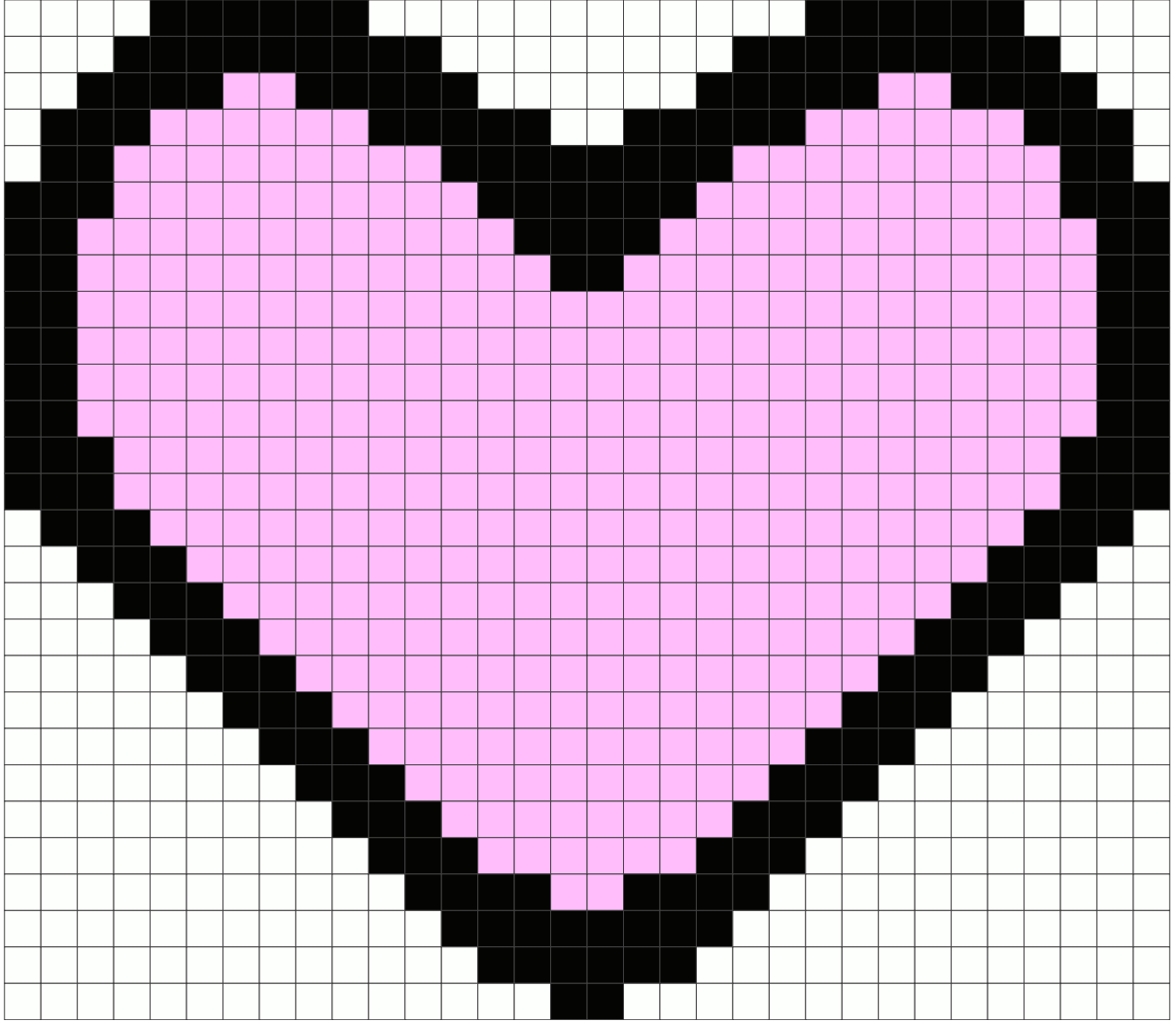Figure 3: A close-up view of Figure 1b.

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

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 $0 \le x \le n-1$ and $0 \le y \le m-1$), 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.
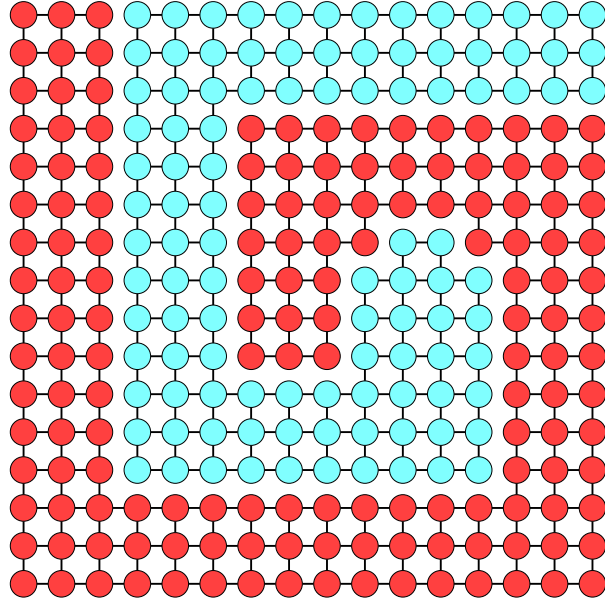
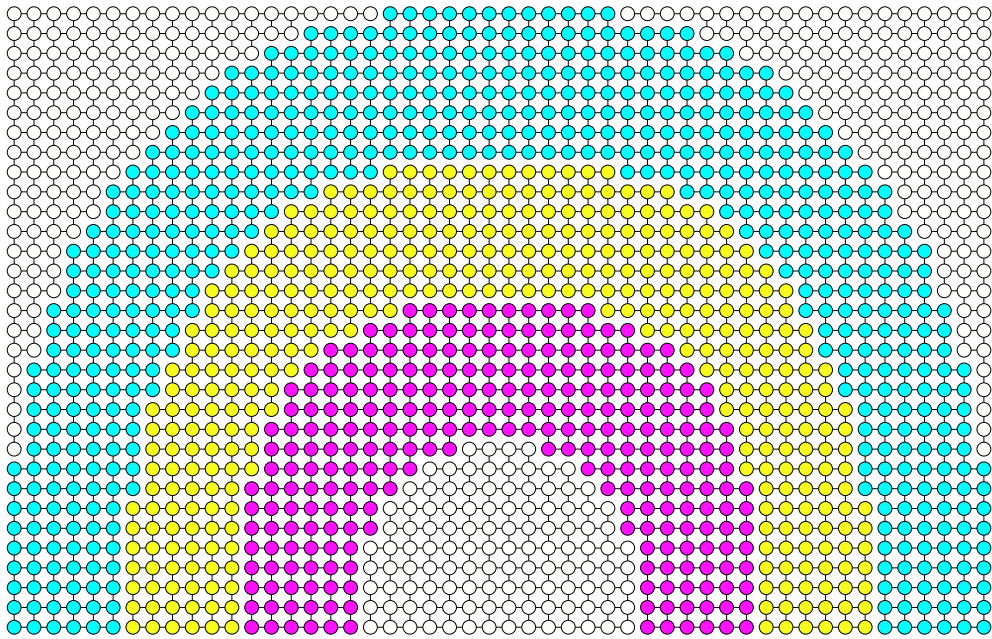Figure 5: The pixel graph for the image in Figure 1a.

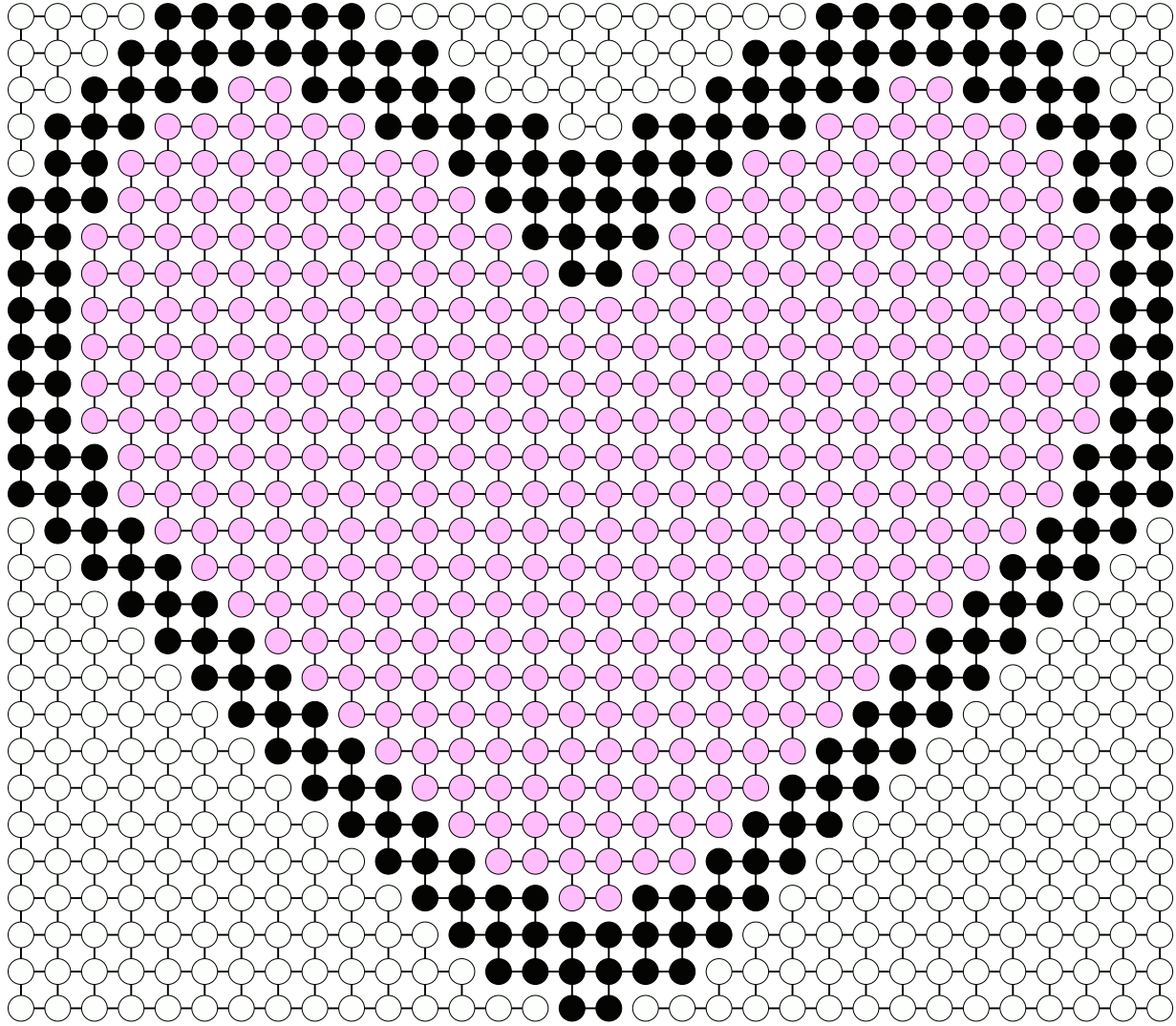Figure 6: The pixel graph for the image in Figure 1b.

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.

2 Programming Assignment

The programming assignment is to implement several graph traversal algorithms using your implementation of the `PixelGraph` data structure from assignment 4. All of the methods to implement are in the file `A5Algorithms.java` (although you are free to modify the `PixelGraph` and `PixelVertex` if necessary).

The template for this assignment is divided among four 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. The templates for `PixelGraph.java` and `PixelVertex.java` are identical to assignment 4, and you should use your assignment 4 implementation as the basis for those data structures (you may add extra features to either class if necessary). 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).

`A5Algorithms.java`	Contains six static methods: `FloodFillDFS`, `FloodFillBFS`, `OutlineRegionDFS`, `OutlineRegionBFS`, `CountComponents` and `A5Bonus` which perform various tasks on a `PixelGraph` instance. Read the comments in the template for more details, and see section 5 below for details on the `A5Bonus` method. **Methods to implement**: `FloodFillDFS`, `FloodFillBFS`, `OutlineRegionDFS`, `OutlineRegionBFS` and `CountComponents`. **Optional Bonus Method**: `A5Bonus`.
`ImageViewerA5.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 `ImageViewerA5.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 `ImageViewerA5.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`. **Use your implementation from Assignment 4**
`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`. **Use your implementation from Assignment 4**

3 Image Viewer Interface

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

```
$ java ImageViewerA5 image_name.png
```

The Java stack size may need to be modified (as in assignment 2) to accommodate the memory demands of a recursive DFS implementation. In that case, you should use the command

```
$ java -Xss64m ImageViewerA5 image_name.png
```

to increase the stack size to 64 megabytes.

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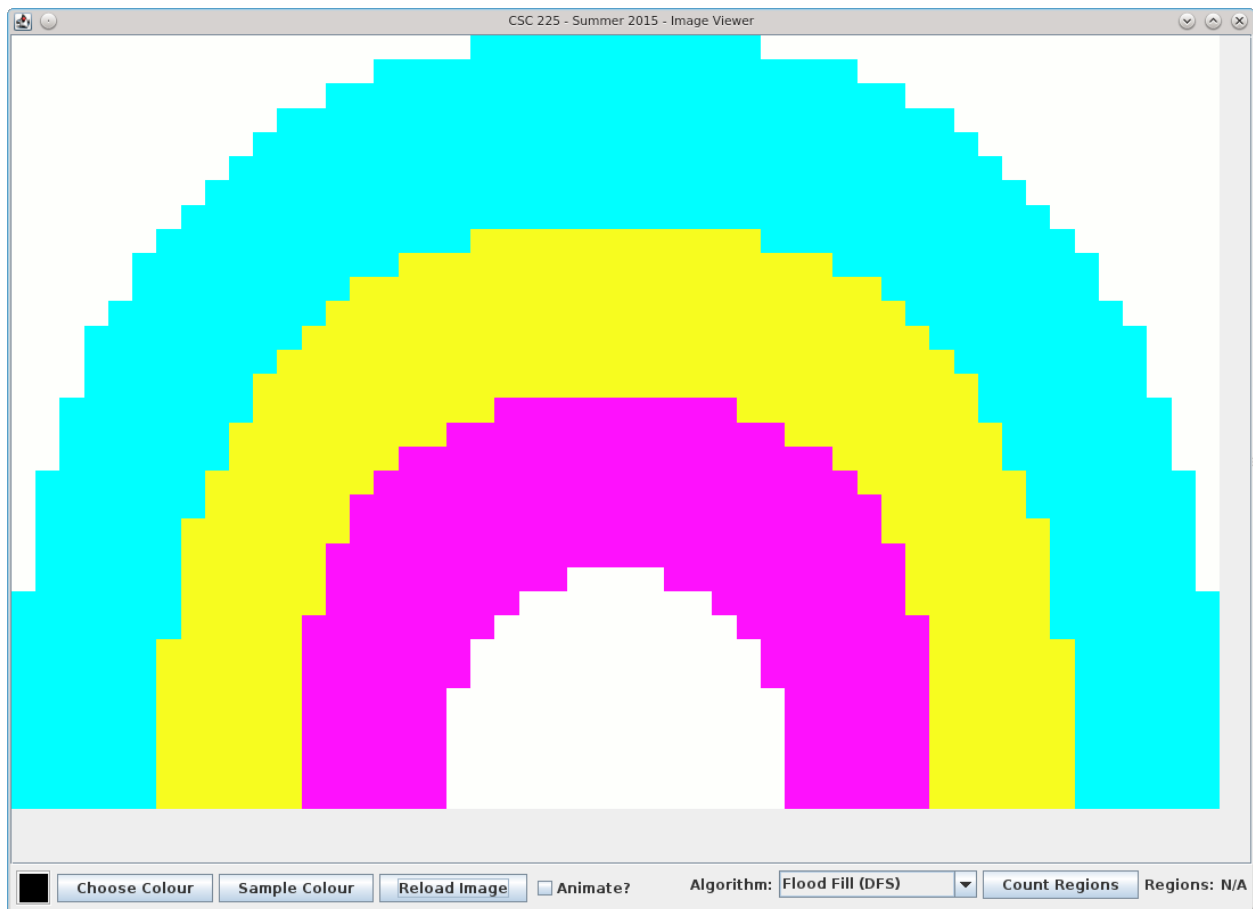

Figure 8: The image viewer window after opening the `rainbow.png` image.

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 `A5Algorithms.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 `A5Algorithms.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 (BFS)" algorithm selected. Note that the DFS and BFS versions of both applications (Flood Fill and Outline Region) should produce identical final results.
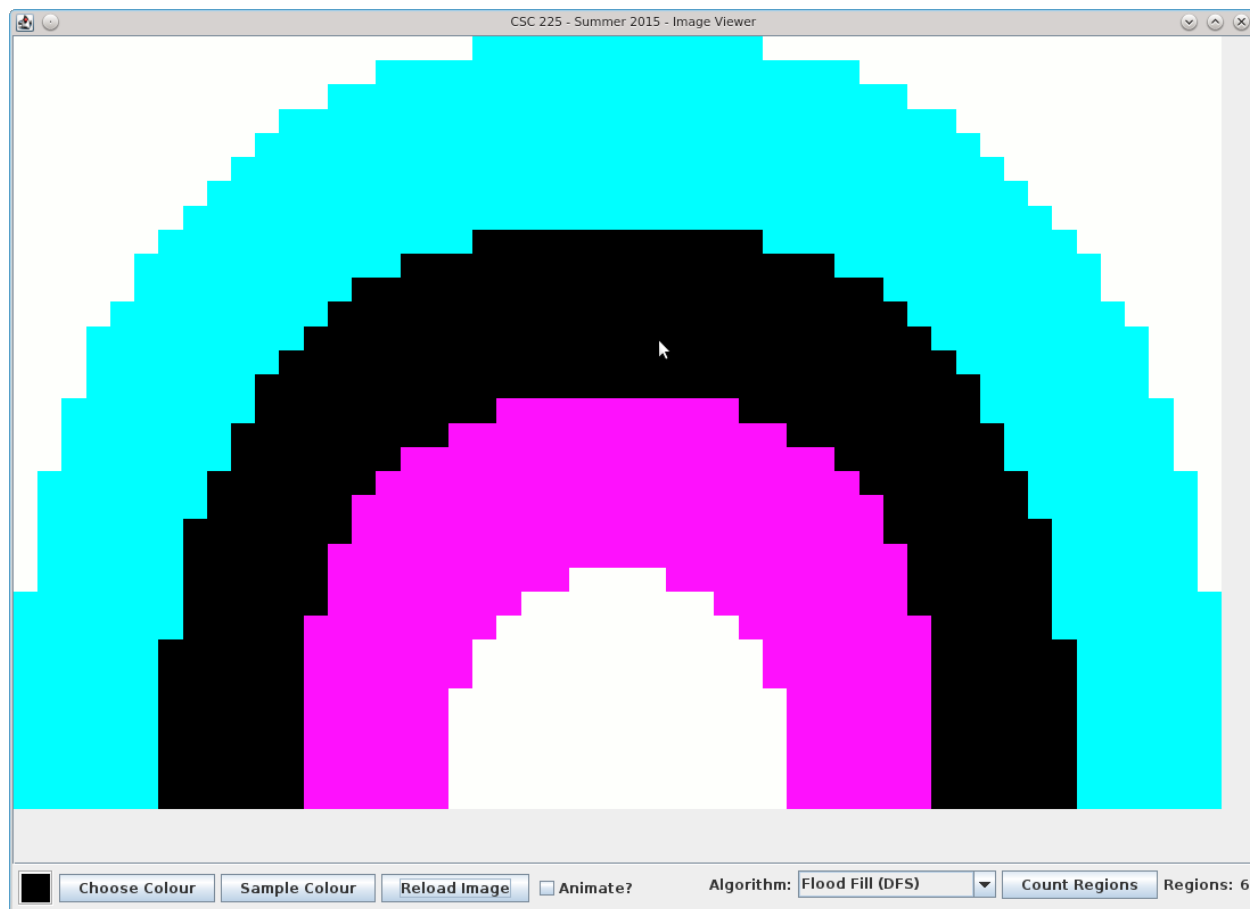

Figure 9: The image viewer window after applying the "Flood Fill (DFS)" algorithm to `rainbow.png` image using black as the fill colour.

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

4   Test Images

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

5   Bonus Mark Opportunity

Selecting the 'Bonus Feature' algorithm in the image viewer results in the empty `A5Bonus` method being executed when the user clicks on a pixel of the image. You may receive up to 5 bonus marks on this assignment by implementing a graph algorithm which performs an interesting image processing task in the `A5Bonus` method. The number of bonus marks awarded will depend on the complexity of the task and its relevance to graph algorithms. Contact the instructor before the due date if you have an idea for a bonus task to determine its eligibility. If you do not contact the instructor about your bonus feature in advance, you may not receive any bonus marks for your implementation.

6   Evaluation Criteria

The programming assignment will be marked out of 40 during a one-on-one demo with an instructor. Each of the methods `FloodFillDFS`, `FloodFillBFS`, `OutlineRegionDFS`, `OutlineRegionBFS` and `CountComponents` will be marked out of 8. During the demo, your code will be tested and inspected, and you may be asked to explain or justify decisions you made during the implementation process. An electronic signup sheet for demo times will be posted on conneX. If you do not sign up for a demo time, or if you sign up but do not attend your demo (without receiving an exception), you will receive a zero on the assignment.

You must submit your assignment electronically through conneX as usual; during your demo, the only version of your code that will be evaluated is the version submitted to conneX.

To be properly tested, every submission must compile correctly as submitted, and must be based on the provided template files. **If your submission does not compile for any reason (even trivial mistakes like typos), or was not based on the provided template files, it will receive at most 5 out of 40**. The best way to make sure your submission is correct is to download it from conneX 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. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. If you do not receive such an email, your submission was not received. If you have problems with the submission process, send an email to the instructor **before** the due date.