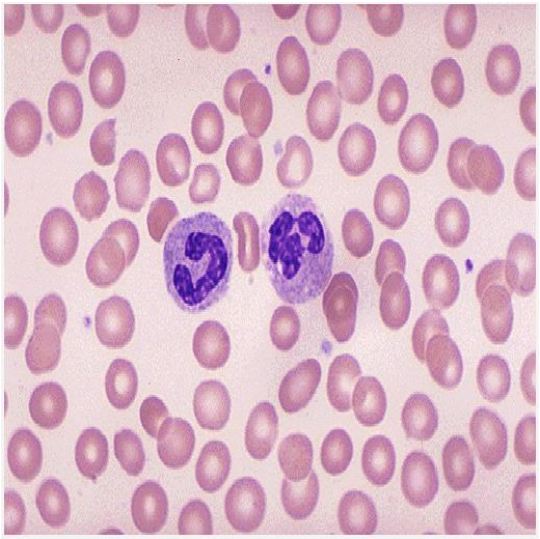
Data Structures and Algorithms 2

CA Exercise 1 – Blood Cell Analyser

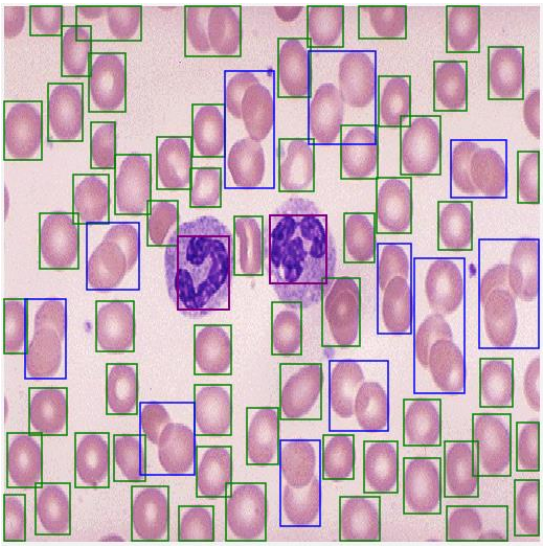
"Create a blood cell analyser system in JavaFX."

The objective of this CA exercise is to create a JavaFX application that can be supplied with an image of a prepared/stained microscopic blood sample showing blood cells. When given an image, the application should automatically locate individual (and clusters of) blood cells and estimate how many of them are in the image overall. It should also differentiate red and white blood cells.

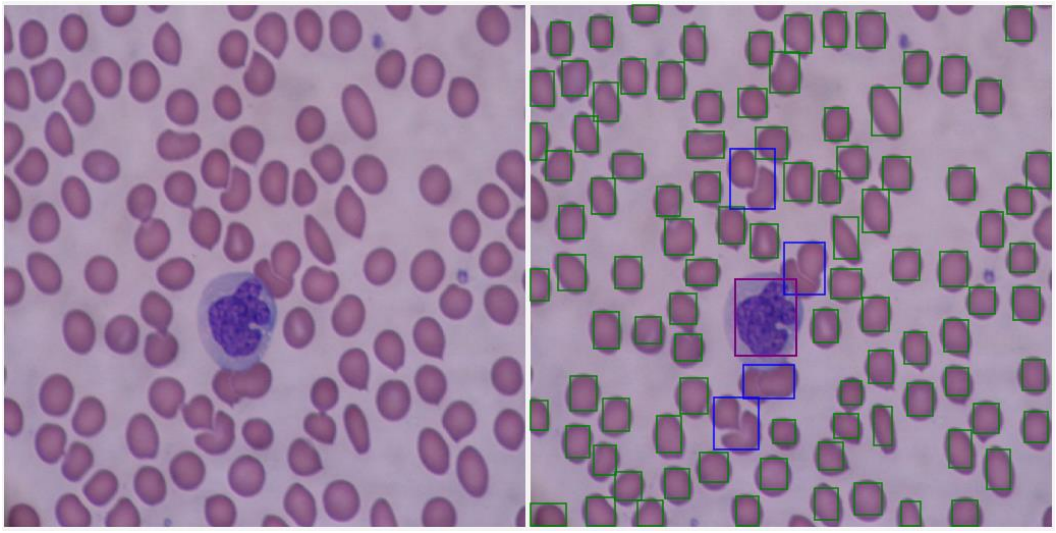
The application should mark the location of individual red blood cells with green rectangles, and clusters of (2 or more) red blood cells with blue rectangles. White blood cells (which are far fewer in number and are differentiated by being larger and having a nucleus) should be marked with purple rectangles. For instance, given the following microscopic blood sample image as input:



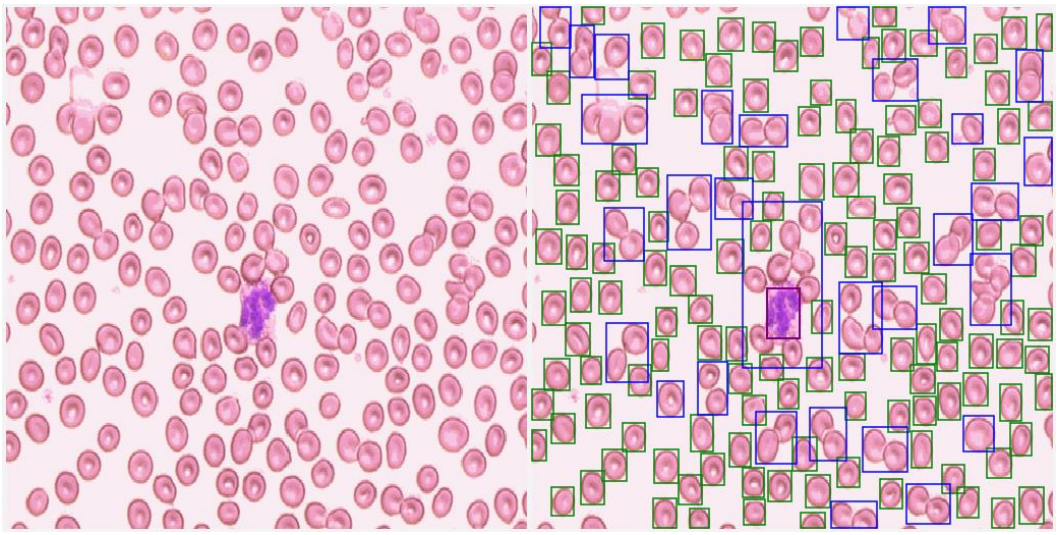
The output might be as follows (estimating 71 red blood cells/clusters and 2 white blood cells):

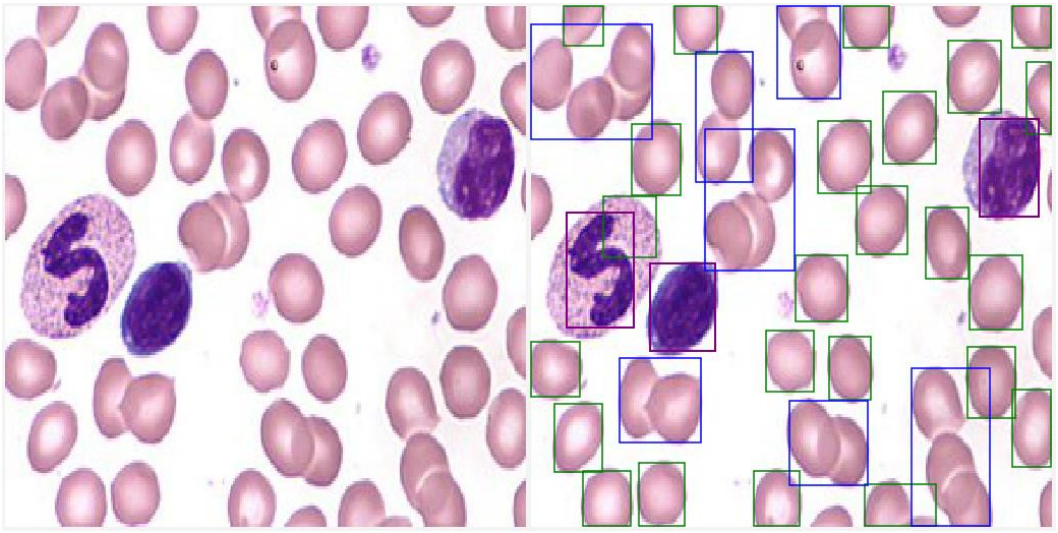


Another example of input/output (99 red blood cells/clusters and 1 white blood cell estimated):



And a couple of other examples:

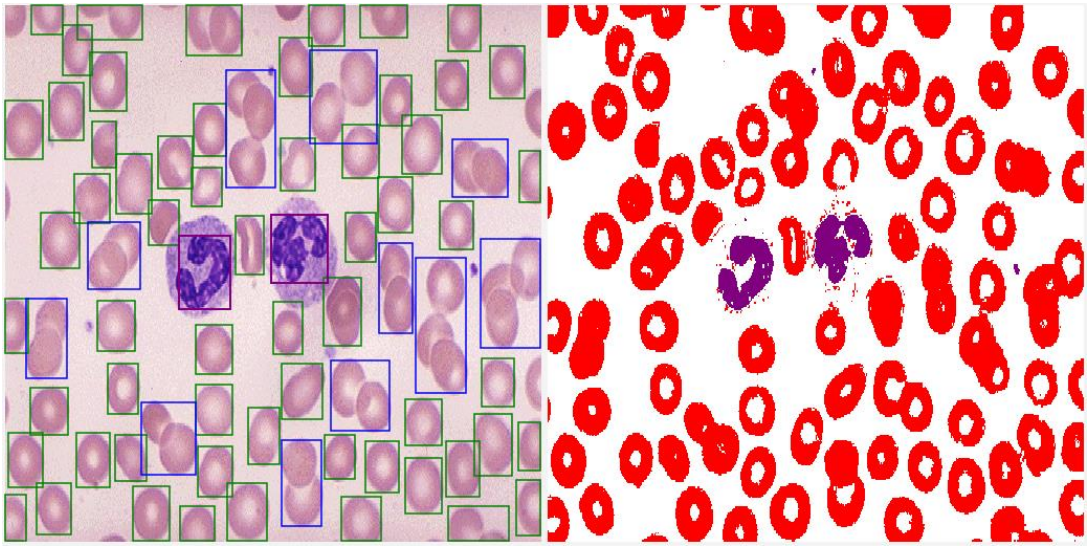


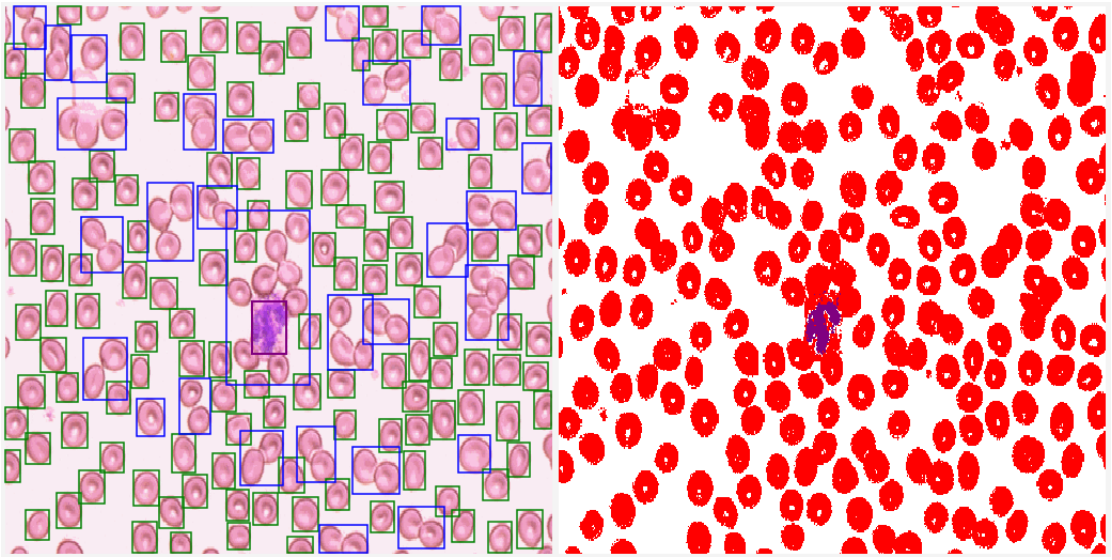


The application should be able to work reasonably well for these and similar scenarios too. It is not expected to work perfectly at all times. For instance, it might incorrectly identify overlapping red blood cells as a single cell instead of a cluster, or a single large red blood cell as a cluster. Partial blood cells in the image (e.g. around the edges) might also cause difficulties. The colourisation, contrast, etc. of the input image will also likely impact how well the system works for it. Some level of image noise reduction/management could be used to improve this (e.g. stripping out outliers/small things unlikely to be blood cells, such as platelets).

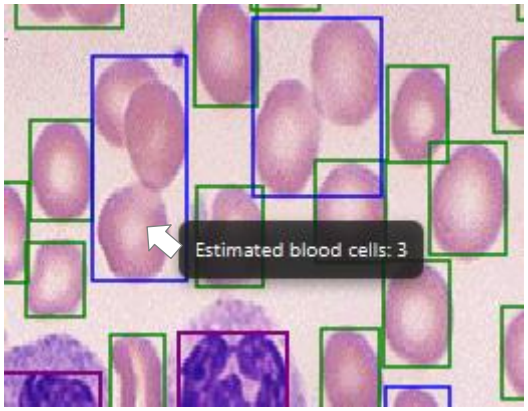
Implementation Notes

- The key aspect of this CA exercise is to use a union-find algorithm to locate the individual blood cells/clusters.
    - Note: you will be dealing with a lot of data (disjoint sets/elements) so remember to use appropriate techniques from last semester to efficiently process large data sets.
- The input image should be converted to tricolour (namely red, purple and white) in the first instance pixel-by-pixel using a suitable luminance/hue/saturation/brightness/RGB calculation.
    - Ideally, pixels belonging to red blood cells should be red, pixels belonging to (the nucleus of) white blood cells should be purple, and all other pixels should be white.
    - If the blood sample has been stained in a typical fashion (Romanowsky staining) you might assume that white blood cells will have a darker purple-blue hue than red blood cells, which will typically be more of a pale pink-grey colour. Different colour channels can possibly be treated differently to aid contrast and noise reduction too. It will likely take a bit of experimentation to get this to work reasonably well.
    - Users should be able to view the tricolour version of the image (e.g. in a separate window, tab, or pane).
    - Some user options/controls might be provided to help the user achieve a good tricolour conversion (e.g. to set hue and saturation thresholds/levels).
    - Tri-colour conversion examples:

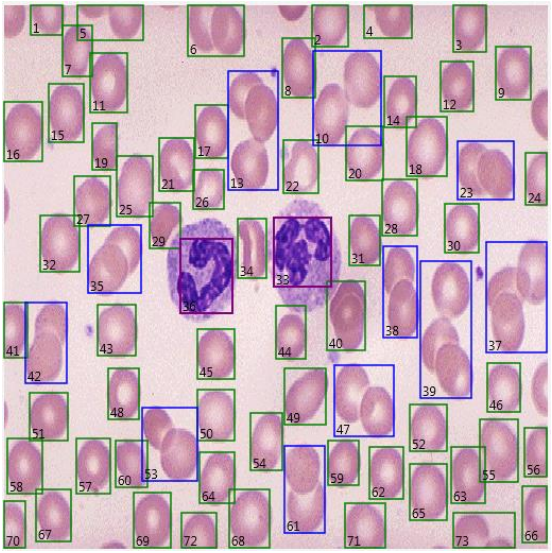


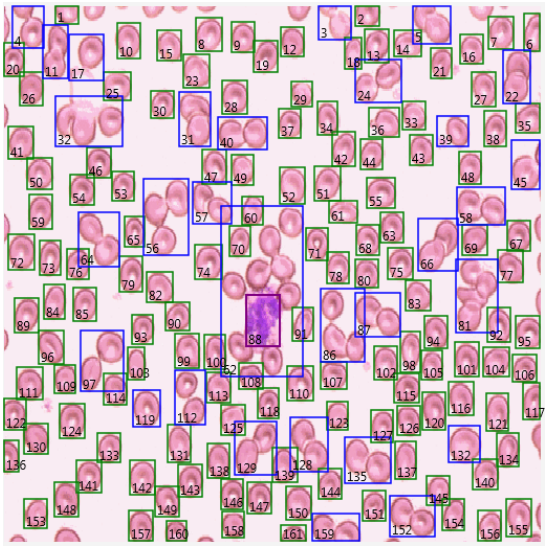


- Each pixel in the tricolour image can initially be considered a disjoint set, with this information potentially represented in an array. Union-find can then be applied to union adjacent red pixels (up, down, left and right) to identify individual red blood cells and clusters of red blood cells (where 2+ blood cells' pixels adjoin, basically). Similarly, adjacent purple pixels can be used to identify white blood cells. White pixels can be disregarded.
    - You may decide to rescale the image from its original size to something smaller for the recognition process (e.g. 512x512, 256x256, 128x128, or similar). A smaller image will require less pixel information to be stored, will be quicker to process, and small "noise" may be (helpfully) lost during the scaling down too.
    - Tip: work with a small test (or fake) image first for development/debugging purposes before using a larger real image.
- Once the union-find is complete, the individual disjoint sets can be processed to identify the set pixel boundaries. The size of the boundary can suggest whether a disjoint set is likely to be an individual blood cell or a cell cluster. Green/blue rectangles based on the boundaries can subsequently be superimposed on the original image to identify red blood cells/clusters. Ditto purple rectangles for white blood cells.
- You should be able to estimate the number of red blood cells within a given cluster too, perhaps allowing users to interactively query it. For instance, touching a cluster with the mouse in the example below:



- A menu option should be provided to allow for each identified blood cell/cluster to be sequentially labelled/numbered starting at 1 if the option is enabled. Examples with the sequential numbering option enabled:





- Some level of image noise reduction and outlier management should be incorporated into the application to allow users to strip out disjoint sets that are heuristically unlikely to be blood cells.
    - Some settings in the application could allow for e.g. minimum and maximum size of blood cells/clusters to be user specified (e.g. in a settings dialog or similar).
    - Consider calculations based on the interquartile range (IQR), for instance, to identify outliers.
    - Some user-adjustable settings (e.g. hue, saturation, etc. threshold levels) could also be provided to achieve a good and clean tricolour image conversion before the union-find operation.
- Overall, an appropriate interactive JavaFX graphical user interface should be provided to allow users to:
    - Select an image file to use.
    - View the original image.
    - Perform and view the tricolour image conversion.
    - Perform the blood cell recognition (and mark blood cells/clusters on the original image, or a copy of it).
    - See estimates of the number of blood cells/clusters in the image.
    - Enable/disable sequential numbering of blood cells/clusters.
    - Adjust settings to manage noise and outliers and to aid the tricolour conversion.
    - Cleanly navigate and exit the application, and have a good user experience overall.
- This is an individual CA exercise and is worth approximately 35% of your overall module mark. You must submit it on Moodle and demonstrate it in the lab for it to be assessed and included in your overall marks. The demo/interview is mandatory!

Indicative Marking Scheme

- Blood sample image file selection and display = 5%
- Tricolour image conversion and display = 10%
- Union-find implementation = 10%
- Onscreen identification of all blood cells/clusters (using rectangles) = 10%
- Distinguishing red and white blood cells (using coloured rectangles) = 10%
- Sequential onscreen blood cell/cluster numbering = 5%
- Estimating/counting of total blood cells/clusters in overall image = 10%
- Estimating/counting of red blood cells within individual clusters = 5%
- Estimating/counting of white blood cells = 5%
- Image noise reduction and outlier management = 10%
- JavaFX GUI = 5%
- JUnit Testing = 5%
- JMH Benchmarking of key methods = 5%
- General (overall completeness, structure, commenting, logic, etc.) = 5%

Note that it is not expected that all students will attempt all aspects of this assignment. Use the above marking scheme to guide your efforts as this is what you will be marked against.