CIS*6020 Assignment 2

Handed out: October 17, 2022 Due: October 31st, 2022

In this assignment you will complete two separate parts:

I. Python code that solves nonogram puzzles (nonogram.py)
II. Python code that converts images into nonogram puzzles (makenonogram.py)

What to hand in:

1. Your **nonogram.py** file

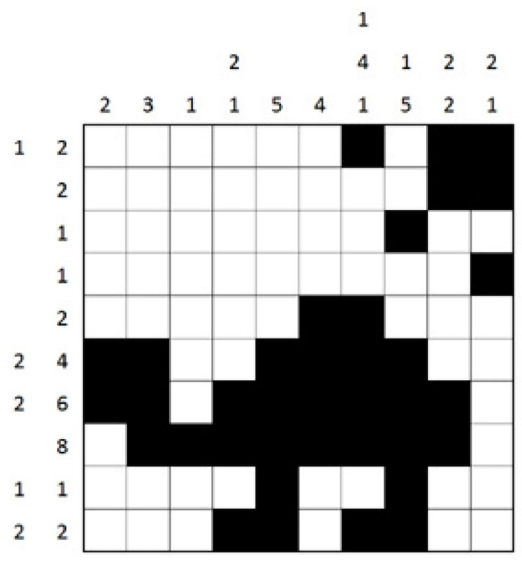
2. Your **makenonogram.py** file

3. A 1 page pdf file named **yourlastname_yourfirstname.pdf** – referenced below in Parts I and II

These should be submitted as a single archived file (e.g. .zip)

Part I:

An example of a nonogram puzzle appears below. The constraints are provided by the numbers on the rows and columns, and indicate in order the "runs" of black values in the grid.



These puzzles are represented in a plain text file format by listing the corresponding numbers for the rows and columns respectively in sequence. Some example files are provided for testing named "example1" "example2" "example3" "example4" and "example5". The case above appears in example4.

You should create a program called nonogram.py that works according to the command:

python nonogram.py example1

You are not provided a framework and should build this from scratch. If you are having trouble, there are many examples on the web of how to design such a solver that vary in complexity. You are allowed to look at these but please use them only as inspiration and do not copy code. A good way of doing this is to look over potential solutions (potentially sketching out a rough workflow), and then close them and work on your own. In your pdf file you should list solutions that you took inspiration from.

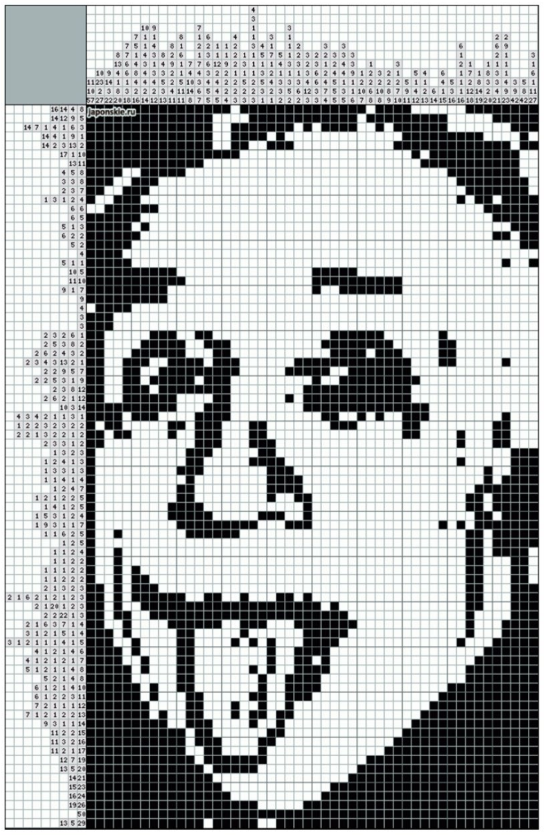
[5 Marks] Your solution can solve puzzles of sizes comparable to example1 and example2 in under 1 minute.

[2 Marks] Your solution can solve puzzles of size comparable to examples 3 and 4 in under 1 minute.

[1 Marks] Your solution can solve puzzles of size comparable to example 5 in under 5 minutes. (Once you get to this scale, things can become much harder) – if you've made significant attempts to solve this, but failed, describe your efforts in the pdf file you submit.

Part II:

For the second part of this assignment, you will fill in the provided makenonogram.py file to create nonogram puzzles with an image provided as input. An example of a sample output puzzle (for a fairly large puzzle) might appear as follows.



While this can be done trivially, the quality of your results will depend on the specific methods that you use. One common operation will involve down-sampling the image from its original size to a more reasonable “puzzle” size that is solvable.

There are two different modes of operation that could be implemented as follows:

1. Threshold based – A color, or gray image can be converted to black and white by assigning pixels above a certain value to white and below the same threshold value to black. Choosing the right threshold can be tricky in practice. There are additional tricks that might be used to improve the appearance using a strategy of this type. (One line of thinking might be to consider what’s done in newspapers)

2. Contour based – An alternative strategy is to detect the edges or contours in the scene, and produce a puzzle that captures the edge boundaries of the scene. In many cases a “mathematical” definition of edges is somewhat different from what humans might draw. Your goal here is to implement an edge based method with the aim of producing output that captures the important characteristics of the input image, or captures important contours that define the form of the input image.

You may use the python toolbox scikit-image (http://scikit-image.org/) to complete this part - you’ll have a hard time without it! The “Gallery” page on the link above will go a long way to providing examples to get you started.

For each case (threshold and contour), you should provide 2 sample images and output puzzles based on images of your choosing. You may choose different images for each method if desired. Make sure to also save a visual representation of the output puzzle.

You should therefore have the following files:

original image 1

original image 2

puzzle image 1

puzzle image 2

text puzzle 1

text puzzle 2

The code should run using the command: python makenonogram.py originalimage1

This should create the puzzle image and the text version for the solver.

Grading:

[7 Marks] Quality and breadth/depth of thresholding or contour based approach

Your sample images will be part of the above judgment – so choose your examples wisely.

Your pdf file should include a brief description (e.g. one paragraph) of how you chose to handle each problem, and what strategies you've employed. You should indicate which of the examples you have been successful at solving. If you have not been successful at completing part I, provide your progress to this point in a description. If you have generated a larger puzzle, there are more efficient solvers around that you might test on – but this isn't a requirement. (It's for your own personal interest)