Assignment 10 (CS141, 2021-2022)

Images are commonly stored as a two-dimensional array containing the pixel values, where each pixel represents a point (dot) on the screen. Pixel values for a black and white image are 0 or 1. For a grey-scale image, pixel values are between 0 and 255. For a coloured image, each pixel is represented as a triple (r, g, b), where $0 \leq r, g, b \leq 255$. For this assignment, we will only deal with grey-scale images.

In this assignment you will be given a grey-scale image as a file and you will edit this image. In particular, you will edit an image given in pgm (portable graymap format) format (https://en.wikipedia.org/wiki/Netpbm#File_formats). Here is a typical image of dimension 4×5 in pgm format. The first line contains the identifier `P2` for pgm images. The second line contains the dimensions $w \times h$ of the image. For the example given below, width $w = 4$ and height $h = 5$. The third line contains the maximum allowed value, which for this assignment, will be 255. Next, the (integer) pixel values for all 4×5 pixels/dots are given.

```
P2
4 5
255
200
167
23
21
2
0
141
98
41
64
190
134
20
165
232
121
26
90
```

```
149
98
```

Now, given an image file of dimension $w \times h$ as input, you will write a program that reads these pixel values into a two-dimensional array (2-D list). Once we have the image data in an array, we will change the values in the array for our editing purpose. When we are done editing (changing the values in the array), we will write data from the array to a new file in the above format such that the file can be opened by an image viewer. This is how our image editor will work!

Our image editor will have two functionalities. One, given an image it can `blur` the image. Second, it can `halve` an image, reduce the dimensions by a factor of two. Let $A[w][h]$ denote the input array (corresponding to the input image). To produce a blurred image, we need to create another array $B[w][h]$ with the same dimensions. One algorithm for blurring is as follows. For any i, j, $B[i][j]$ equals the average of $A[i-1][j], A[i+1][j], A[i][j-1], A[i][j+1], A[i][j]$. In order to `halve` an image, we create an array $B[w/2][h/2]$ of dimensions $\lceil w/2 \rceil$ and $\lceil h/2 \rceil$. For any i, j, $B[i][j]$ is the average of $A[2i][2j], A[2i][2j+1], A[2i+1][2j], A[2i+1][2j+1]$. If for pixels at corner points, some of these indices become negative, we will ignore those and only take into account pixel values for non-negative indices.

Now, let us look at how to write this program. First, we open the input file in the read mode, and read the contents line by line. We may use the readline() function for this purpose. The first time readline() is executed on the sample input given as above, it returns "P2". The second time it is run, it returns "4 5" as a string. We need to extract the individual dimensions $w = 4$ and $h = 5$ from this. You may use the split() function for this purpose. If s is a string of many words, $s.split()$ returns a list of words. You may access the integer values for w and h by indexing into appropriate locations in the list. Once you retrieve the dimensions of the image as mentioned above, you read the pixel values into a two-dimensional array. Next, you create a new two-dimensional array and modify its values. Once you are done reading from the file, you should close the file. This is optional if you are reading using the `with` function.

Sample code to read from file "input.txt" in read mode.

```
f = open ("./input.txt", 'r') # Store 'input.txt' in the current directory.
firstline = f.readline()
f.close()
```

Once you have the output two-dimensional array ready, create an output file `output-image.pgm` by executing a file open function in the write mode. Now, we want to write values to this file such that it is interpreted as a pgm image file. First, we write `P2` to this file. Then, we write the dimensions of the image and the maximum pixel value. Next, we write the pixel values of individual pixels or dots. You may use the file `write` function for this purposes. Once you are done writing these values, you should close the file.

Sample code to write to file "output.txt" in write mode.

```
f = open ("./output.txt", 'w') # 'output.txt' will be in the current directory.
f.write("what do you want to write?\n")
f.close()
```

You are done now, and enjoy the edited image in an image viewer.