C PROGRAMMING

Collaboration Policy: You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in a comment at the top of your files. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite by naming the book etc. or listing a website's URL. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: List computing ids in comments at the top of your file

Sources: List URLs etc. in comments at the top of your file

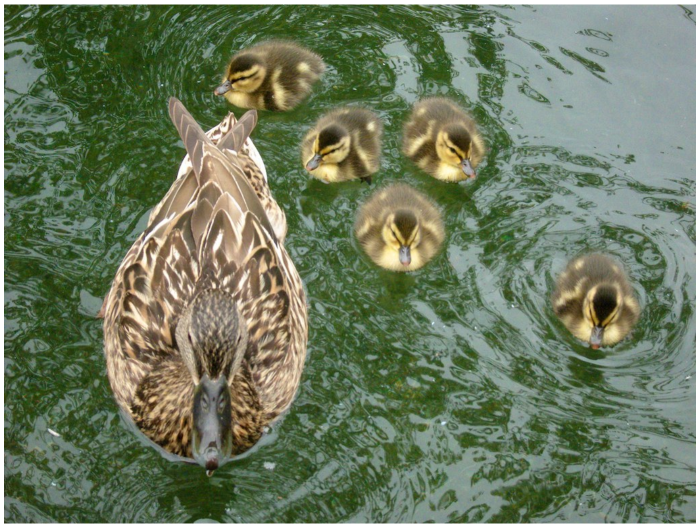

Figure 1: Ducks

PROBLEM 1 *Seam Carving*

George Costanza needs our help! In the summer of 1989, after the "boombox incident" on the beach, he finds that he is in the background of his new boss' family photo. The technology of the time, airbrushing, caused him some trouble, but now there's Photoshop, Gimp, and more importantly, **seam carving**! Before we move on to George's slightly more complicated version of the problem, let's start with the ducks pictured in Figure 1. We want to remove the "less interesting" parts of the picture (the water) to bring the ducklings closer to their mother. You must write a **dynamic programming** algorithm that finds the lowest-energy "seam" in the image.

You will be given a color picture consisting of an $n \times m$ array `image[0..n-1][0..m-1]` of pixels as seen in Figure 2, where each pixel specifies a triple of red, green, and blue (RGB) intensities. The intensity of a color is an integer value between 0 and 255. Specifically, accessing the red intensity of pixel $p_{i,j}$ (the pixel in image column i and row j) would be `image[j][i][0]`, the green intensity would be accessed by `image[j][i][1]`, and the blue intensity accessed by `image[j][i][2]`. Note: for images, the $(0,0)$ position is the top-left corner of the image.

To move our ducks closer, we wish to remove one pixel from each of the n rows of the image, so that the whole picture becomes one pixel narrower. But we also want to avoid any disturbing visual effects, so we require that the pixels removed in two adjacent rows be in the same or adjacent columns; therefore the pixels removed form a "seam" from the top row to the bottom row where successive pixels in the seam are adjacent vertically or diagonally.

We also want to remove the less interesting parts of the picture, so we also want to calculate the real-valued energy measure $e(p_{i,j})$ for each pixel in our image. We define this measure as the average of how different a pixel's colors are from its eight neighboring pixels (i.e., the pixels immediately adjacent to the top, left, right, and below, as well as the four diagonally-adjacent ones). Specifically, the energy of pixel $p_{i,j}$ is defined as

$$e(p_{i,j}) = \frac{1}{8}\sum_{x=i-1}^{i+1}\sum_{y=j-1}^{j+1} d(p_{i,j}, p_{x,y}),$$

where the difference in pixel colors is defined by their 3D Euclidian distance,

$$d(p_1, p_2) = \sqrt{(p_2[\text{red}] - p_1[\text{red}])^2 + (p_2[\text{blue}] - p_1[\text{blue}])^2 + (p_2[\text{green}] - p_1[\text{green}])^2}.$$

Note that by definition, $d(p_{i,j}, p_{i,j}) = 0$, so the energy function is indeed computing the average difference between a pixel's colors and those of its eight neighbors. If a pixel is on the edge of the image, calculate the average of differences between its colors and those of its available neighbors. Intuitively, the lower a pixel's energy measure, the more similar the pixel is to its neighbors. We define the energy measure of a seam to be the sum of the energy measures of its pixels. Your algorithm **must** find a seam with the lowest energy measure.

For this assignment, you are provided scaffolding code that reads in an image and converts it to a 3D array of pixel color information. It then passes that array to the `SeamCarving` class, which you will implement, to find the lowest-weight seam. Finally, it prints out the weight of the seam, the ordered x-coordinates of the seam, and has a commented-out section to highlight and graphically display the seam for testing purposes. You must implement the `run()` method that accepts the 3D `image` array and returns the weight of the lowest-weight seam (as a double-precision number). Your method must also use backtracking to store the seam as an ordered list of x-coordinates (from top to bottom of the image) as an instance variable to be returned by the `getSeam()` getter method that you must also implement. That method should return an array of integers.

Additional Details

- Your algorithm must be written in Python (3.6) or Java (11).

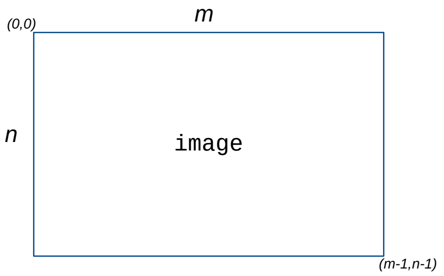

Figure 2: Our images are stored in `image[j][i][c]`, where i is the x-coordinate of the image and j is the y-coordinate of the image. The $(0,0)$ coordinate is the top-left of the image. At each (i,j) position, there is a triple ($c \in \{0,1,2\}$) of color intensities. `image[j][i][0]` is the red intensity, `image[j][i][1]` is the green intensity, and `image[j][i][2]` is the blue intensity.

- You must download the appropriate wrapper code from Collab based on the language you choose: `main.py` and `seam_carving.py` for Python, `Main.java` and `SeamCarving.java` for Java.
- Implement the `run()` and `getSeam()` methods in the `SeamCarving` class. The `run()` method should execute the entirety of your algorithm and store the seam and seam weight as instance variables; `getSeam()` should return the resulting seam without additional computation (i.e. it should be a "getter").
- For Python, you will need the `PIL` library. Information on this library can be found at `https://pillow.readthedocs.io/en/stable/installation.html`.
- Two test images are provided, along with the picture of the ducks above. You should produce additional tests, including edge cases.
- You *may* modify the `Main.java` or `main.py` files to test your algorithm, but they **will not** be used during grading.
- You must submit your `SeamCarving.java` or `seam_carving.py` files on Collab. Do not zip them. Do **not** submit `Main.java`, `main.py`, or any test files.
- A few other notes:
  - Your code will be run as:
    `python3 main.py` for Python,
    or `javac *.java && java Main` for Java.
  - You may upload multiple Java files if you need additional classes, but do not assign packages to the files

Since we want to eventually use this algorithm to help out George, we want it to be efficient. Therefore, your algorithm **must** run in $O(nm)$ time.

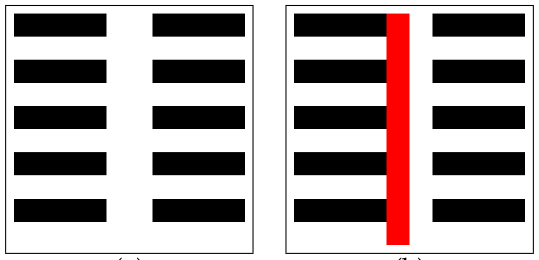

Figure 3: (a) Sample input image *test1.png*. (b) A lowest-weight seam colored red.

```
weight: 839.1786162671211
seam: [4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
time: 0.004
```

Figure 4: The text-based output from the *main.py* or *Main.java* file for the example image in Figure 3.

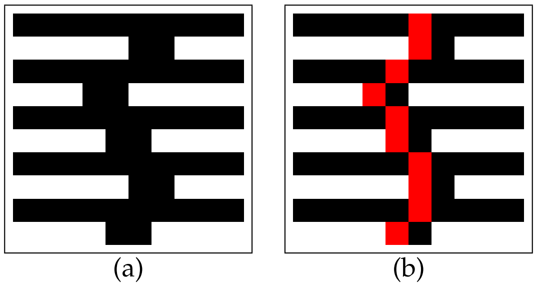
Figure 5: (a) Sample input image *test2.png*. (b) A lowest-weight seam colored red. Note that for this input image,

```
weight: 894.3877357583791
seam: [5, 5, 4, 3, 4, 4, 5, 5, 5, 4]
time: 0.004
```

Figure 6: The text-based output from the *main.py* or *Main.java* file for the example image in Figure 5.

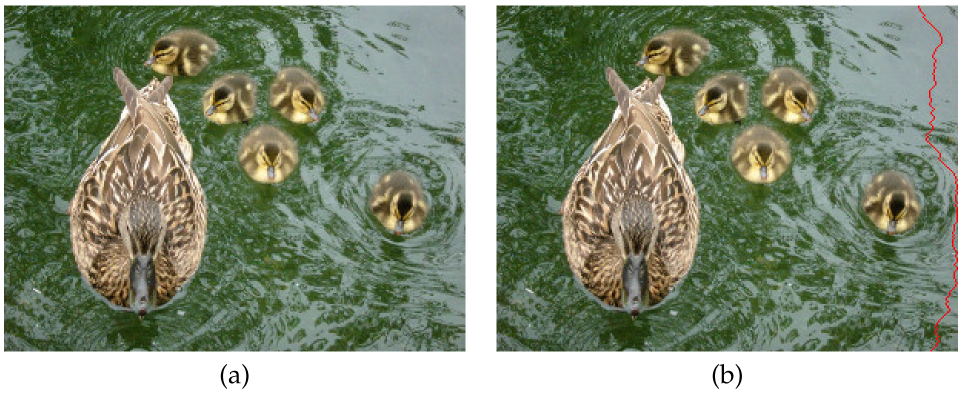
Figure 7: (a) Our sample image with of ducks. (b) A lowest-weight seam colored red. Note that we would need to repeatedly find and remove least-weight seams to move our ducks closer, but for the assignment we will only be finding the first seam.

```
weight: 2084.8176202077766
seam: [274, 274, 275, 275, 276, 277, 278, 278, 278, 279, 280, 281, 281, 280,
279, 278, 278, 277, 278, 277, 276, 275, 275, 275, 275, 275, 276, 277, 278,
279, 280, 281, 282, 282, 283, 284, 285, 284, 284, 284, 285, 286, 285, 284,
283, 284, 284, 284, 285, 285, 285, 285, 284, 283, 282, 281, 281, 281, 280,
281, 281, 282, 281, 281, 282, 281, 282, 283, 282, 282, 283, 284, 285, 284,
284, 285, 284, 283, 283, 284, 283, 282, 281, 280, 279, 279, 278, 278, 279,
280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 288, 288, 289, 290, 291,
292, 293, 294, 295, 296, 297, 297, 297, 298, 299, 299, 298, 299, 299, 299,
299, 299, 299, 299, 299, 299, 298, 298, 298, 299, 298, 299, 299, 299, 299,
299, 299, 299, 299, 299, 299, 299, 298, 297, 297, 297, 297, 297, 296, 297,
297, 296, 297, 296, 296, 295, 296, 296, 296, 297, 298, 299, 299, 299, 298,
299, 299, 299, 299, 298, 299, 298, 299, 299, 299, 298, 297, 297, 297, 297,
298, 299, 299, 298, 297, 296, 295, 294, 293, 294, 293, 292, 293, 294, 295,
296, 297, 296, 295, 294, 293, 292, 291, 290, 289, 288, 287, 286, 285, 284,
283, 283, 282, 281, 282, 283, 284, 285, 286, 286, 285, 286, 285, 284, 284,
283]
time: 0.012
```

Figure 8: The text-based output from the *main.py* or *Main.java* file for the example image in Figure 7.