Programming Assignment 7: Seam Carving

Frequently Asked Questions

How do I manipulate images in Java? Use our Picture data type (which is part of `stdlib.jar`) and the Color data type (which is part of the java.awt library). Here is some more information about theColor and Picture data types. Luminance.java and Grayscale.javaare example clients that use the `Color` and `Picture` data types, respectively.

I noticed that the Picture API has a method to change the origin (0, 0) from the upper left to the lower left. Can I assume (0, 0) is the upper left pixel? Yes.

The data type is mutable. Does this mean I don't have to make a defensive copy of the Picture? No, you should still make a defensive copy of the picture that is passed to the constructor because your data type should not have any side effects (unless they are specified in the API).

Why does the energy function wrap around the image? This is one of several reasonable conventions. Here are a few advantages of defining it in this way:

- If the color of the pixels on one side of the picture are the same as on the opposite side, then these pixels probably represent the background (and a less interesting feature of the image).
- It reduces the likelihood of getting ties when finding the minimum energy seam, say, as opposed to defining the energy of every border pixel to equal some large number.
- It reduces the number of corner cases.

Why can't seams wrap around the image (i.e., why not treat the image as a torus when computing seams)? While it would be consistent with the way that we define the energy of a pixel, it often produces undesirable visual artifacts. (The API from Spring 2014 allowed seams to wrap around, which explains why in Josh Hug's video they do.)

Identifying the shortest energy path in a picture looks a lot like the COS 126 dynamic programming assignment about genetic sequencing. Can I use that approach? You are welcome to use a dynamic programming approach; however, in this case, it is equivalent to the topological sort algorithm for finding shortest paths in DAGs.

My program is using recursion to find the shortest energy path. Is this ok? You should not need recursion. Note that the underlying DAG has such special structure that you don't need to compute its topological order explicitly.

My program seems to be really slow. Any advice? We strongly encourage you to implement these optimizations.

- Call `energy()` and `get()` only once per pixel.
- Keep your instance variables to a minimum. This goes hand in hand with deferring any calculations until necessary.

- Don't use an explicit `EdgeWeightedDigraph`. Instead, execute the topological sort algorithm directly on the pixels.
- The order in which you traverse the pixels (row-major order vs. column-major order) can make a big difference.

What values of W and H should I use when timing? The value of W and H are not as significant as the resulting time. Be sure that the resulting time is greater than 1 second for all your data.

How do I determine the relationship between W and H? Analyze the code. Use the feedback you got on the Percolation assignment to help you.

Testing

Clients. You may use the following client programs to test and debug your code.

- PrintEnergy.java computes and prints a table of the energy values of the image whose name is specified as a command-line argument.
- ShowEnergy.java computes and draws the energy of the image whose name is specified as a command-line argument.
- ShowSeams.java computes the horizontal seam, vertical seam, and energy of the image whose name is specified as a command-line argument. It draws the horizontal and vertical seams over the energy.
- PrintSeams.java computes the horizontal seam, vertical seam, and energy of the image whose name is specified as a command-line argument. It prints square braces around the energies of the horizontal and vertical seams. Many of the small input files provided also have a `printseams.txt` file (such as 5x6.printseams.txt), so you can compare your results to the correct solution.
- ResizeDemo.java uses your seam removal methods to resize the image. The command-line arguments are the image filename, *r*, and *s*, where *r* is the number of columns and *s* is the number rows to remove from the image specified. This file works best with real images, not small test files.
- SCUtility.java is a utility program used by some of the above clients.

Sample input files. The directory seamCarving contains these client programs above along with some sample image files. You can also use your own image files for testing and entertainment.

Your code should work no matter the order or how many times the methods are called. A good test would be to find and remove a seam, but then make sure that finding and removing another seam also works. Try all combinations of this with horizontal and vertical seams.

Possible Progress Steps

These are purely suggestions for how you might make progress. You do not have to follow these steps.

1. Start by writing the constructor as well as `picture()`, `width()`, and `height()`. These should be easy. Be sure to defensively copy the `Picture` before returning to the client. Why?

2. Next, write `energy()`. Calculating Δ_x^2 and Δ_y^2 are very similar. A private method which can calculate either one will keep your code simple. To test that your code works, use the client `PrintEnergy` described in the testing section above.

3. To write `findVerticalSeam()`, you will want to first make sure you understand the topological sort algorithm for computing a shortest path in a DAG. It is recommended that you do *not* create an `EdgeWeightedDigraph`, as the resulting code will be not only slower, but also harder to write. Instead, construct a *W*-by-*H* energy matrix using the `energy()` method that you have already written. Think about which data structures (in addition to the 2D energy matrix) that you will need to implement the shortest path algorithm. To test, use the client `PrintSeams` described in the testing section above.

4. Now implement `removeVerticalSeam()`. Typically, it will be called with the output of `findVerticalSeam()`, but be sure that it works for *any* valid seam. To test, use the client `ResizeDemo`described in the testing section above.

5. To implement `findHorizontalSeam()` and `removeHorizontalSeam()`, *transpose* the picture and call `findVerticalSeam()` and `removeVerticalSeam()`. Don't forget to transpose the picture back, when needed.

A video is provided for those wishing additional assistance. Warning: the video was made in early 2014 and is somewhat out of date. For example the API has changed.