FREQUENTLY ASKED QUESTIONS (GENERAL)

How do I manipulate images in Java? Use our Picture data type (which is part of `algs4.jar`) and Java's java.awt.Color data type. Luminance.java and Grayscale.java are example clients.

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 (and you should not call this method).

FREQUENTLY ASKED QUESTIONS (SEAMCARVER)

Must the arguments to `removeHorizontalSeam()` and `removeVerticalSeam()` be minimum energy seams? No. Each method should work for any valid seam (and throw an exception for any invalid one).

The `SeamCarver` data type is mutable. Must I still defensively copy of the `Picture` objects? A data type should not have side effects (unless they are specified in the API). It should behave properly as prescribed even if the client mutates the `Picture` object passed to the constructor or returned from the `picture()` method. As a result, you might need to make defensive copies of any `Picture` objects that you either take as input or return as output.

Can I use dynamic programming? Yes, though, in this context, dynamic programming 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 okay? 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 recommend that you implement the first four of these optimizations, but you can improve performance further my implementing some of the later ones.

- Don't use an explicit `EdgeWeightedDigraph`. Instead, execute the topological sort algorithm directly on the pixels.
- When finding a seam, call `energy()` at most once per pixel. For example, you can save the energies in a local variable `energy[][]` and access the information directly from the 2D array (instead of recomputing from scratch).
- Avoid redundant calls to the `get()` method in `Picture`. For example, to access the red, green, and blue components of a pixel, call `get()` only once (and not three times).
- The order in which you traverse the pixels (row-major order vs. column-major order) can make a big difference.

- Creating `Color` objects can be a bottleneck. Each call to the `get()` method in `Picture` creates a new `Color` object. You can avoid this overhead by using the `getRGB()` method in `Picture`, which returns the color, encoded as a 32-bit `int` . The companion `setRGB()` method sets the color of a given pixel using a 32-bit `int` to encode the color.

- Don't explicitly transpose the `Picture` or `int[][]` until you need to do so. For example, if you perform a sequence of 50 consecutive horizontal seam removals, you should need only two transposes (not 100).

- Consider using `System.arraycopy()` to shift elements within an array.

- Reuse the energy array and shift array elements to plug the holes left from the seam that was just removed. You will need to recalculate the energies for the pixels along the seam that was just removed, but no other energies will change.

How much memory can my `SeamCarver` object use as a function of *W* and *H*? A `Picture` objects uses ~ 4*W H* bytes of memory—a single 32-bit `int` per pixel. Unless you are optimizing your program by updating only the energy values that change after removing a seam, you should not need to maintain the energy values in an instance variable. Similarly, the `distTo[][]` and `edgeTo[][]` arrays should be local variables, not instance variables. For reference, a `Color` object consumes 48 bytes of memory.

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 of an image with filename provided on the command line.

- ShowEnergy.java computes and draws the energy of an image with filename provided on the command line.

- ShowSeams.java computes the horizontal seam, vertical seam, and energy of the image with filename provided on the command line. Draws the horizontal and vertical seams over the energy.

- PrintSeams.java computes the horizontal seam, vertical seam, and energy of the image with filename provided on the command line. Prints the horizontal and vertical seams as annotations to the energy. 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 filename, *W* and *H* where *W* is the number of columns and *H* is the number rows to remove from the image specified.

- SCUtility.java is a utility program used by most of the above clients.

Sample input files. The zip file seam.zip contains the client programs above along with some sample image files. You can also use your own image files for testing and entertainment.

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 very easy.

2. From there, write `energy()`. Calculating Δ_x^2 and Δ_y^2 are very similar. Using two private methods 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 topologial sort algorithm for computing a shortest path in a DAG. Do *not* create an `EdgeWeightedDigraph`. Instead construct a 2d energy array using the `energy()` method that you have already written. Your algorithm can traverse this matrix treating some select entries as reachable from `(x, y)` to calculate where the seam is located. To test that your code works, use the client `PrintSeams` described in the testing section above.

4. To write `findHorizontalSeam()`, transpose the image, call `findVerticalSeam()`, and transpose it back.

5. Now implement `removeVerticalSeam()`. Typically, this method will be called with the output of `findVerticalSeam()`, but be sure that they work for any seam. To test that your code words, use the client `ResizeDemo` described in the testing section above.

6. To write `removeHorizontalSeam()`, transpose the image, call `removeVerticalSeam()`, and transpose it back.