Overview:

The task this time was to produce a program that could take an image and resize it using seam carving to dimensions entered by the user. In contrast to the last project there was a lot less code writing, so it seemed, however there was a lot more thought involved to produce the required code. In more specific terms this project required the user to create classes that could: calculate the energy of a pixel from its RGB values and the RGB values of the surrounding pixels, find the lowest valued contiguous seam of pixels, and be able to remove a seam of pixels from an image and output the new seam. This task was quite challenging as, unlike the last project, we had to work with libraries we had not used before. In terms of extended work, we implemented a GUI which was not specified as necessary, it includes a file chooser. As an extension we have also ensured that cropping does not affect one side or end of the image more than another (if there are seams of equal length on either side of the image). Note that, since our extensions don't overhaul the functioning of the main project, we decided to include them as part of the main project.

Build Instructions:

As mentioned above this project comes with a GUI so all that is needed is compiling of all the code in the source folder and then from the source folder running:

Java GUI

Use the intuitive design to choose an image and enter the height and width.

Design & Implementation:

For this project we created many classes for our solution. This was due to us trying to use as little libraries as possible for our own benefit of being creative and not relying on them. This section will be decomposed into a brief top-level description and then step specific descriptions of the class

Top-Level:

The GUI classes are generally self-explanatory in their purpose, the GUI class is the main GUI and the image finder is a secondary interface that allows the user to choose an image form their files. The calculation of each pixel's energy is done using the energy calculation class and the pixel class. The identification of the lowest energy seam is found using the memorized index and the seam identifier class'. The seam removal is handled by the seam carver class. Beyond that the procedure of method calls to get the desired output and the validation of the input is mainly done in the GUI class.

Energy Calculation:

The Energy Calculation methods are used in this program to calculate a so-called 'energy' of each pixel, which is a score of its importance in relation to its surrounding pixels. The calculation of the energy of a pixel is as specified in the practical specification: First, the difference between the RGB values of the adjacent pixels are calculated (separately in the x- and y- directions). A root mean square (rms, referring to squaring all the values, adding them, and then square-rooting) average of the differences between the individual RGB values of a target pixel's adjacent pixels is then calculated for the x-adjacent and y-adjacent pixels, and then an rms average of these values gives the energy of a target pixel. To help calculate the energy calculation, I decided to create a new class called Pixel, which stores three ints, red, green and blue, which store the separated RGB values of the corresponding pixel, and a boolean representing whether or not the pixel is part of the lowest-weight seam as calculated in the Seam Identification section. Then I recreated the image as a 2D

array of Pixel objects, and converted each image to their component RGB values.This implementation also helps with the seam removal implementation.

Seam Identification:

The purpose of the Memoized Index class is to correspond to a pixel in the energy matrix from the image. Each Memoized Index represents a value in a seam, it holds the total value of energy so far as well as referencing the next pixel in that seam. The reason for this is so that we can have a structure that will help dynamically program. Each memoized index stores the calculations made so far in any given seam and this means we don't have to make repeat calculations and allows us to make a non-greedy algorithm (at the cost of storage). The seam identifier calculates the lowest energy seam a pixel can access and then creates a new memoized index, referencing the next pixel in the seam and setting its energy to the corresponding energy from the image plus the rest of the seam. The avoidCrop boolean allows equal seams to be taken from either side of the image, which ensures equal cropping/seam removal from across the image

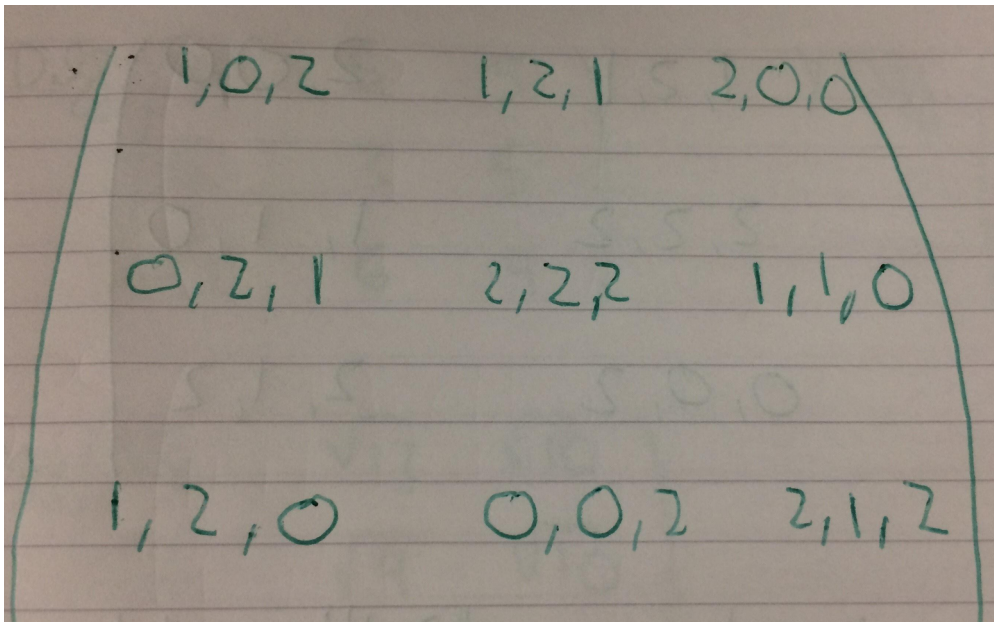
Seam Removal:

The seam removal class is the final step in the main function of the program; to take an image with a lowest weight seam of pixels and remove said seam from the image, and finally rebuilding the image, and also to repeat this as many times as is needed for a user to reduce the dimensions of an image. To do this, I first 'marked' all of the pixels in the calculated seam by changing the boolean IsLowestEnergy, one of the attributes of the Pixel class, to true for all the pixels in the previously mentioned 2D array that are in the seam. The method then iterates through the 2D array, replicating the unmarked pixels in a new array which is one index shorter in the corresponding direction (whether a vertical or horizontal seam is being removed), and not adding the marked pixels to the new array. While doing this it also recreates a new image much in the same way. These are stored for later in case we want to remove more than one seam. After this the seam-removed image can be either output or used as the new image image for which a seam needs to be removed again. An important note is that separate methods are required for removing horizontal and vertical seams because, for example, when we are removing a horizontal seam, each column of pixels will uniformly have one less pixel whereas some rows of pixels may have many removed pixels and some may have none removed.

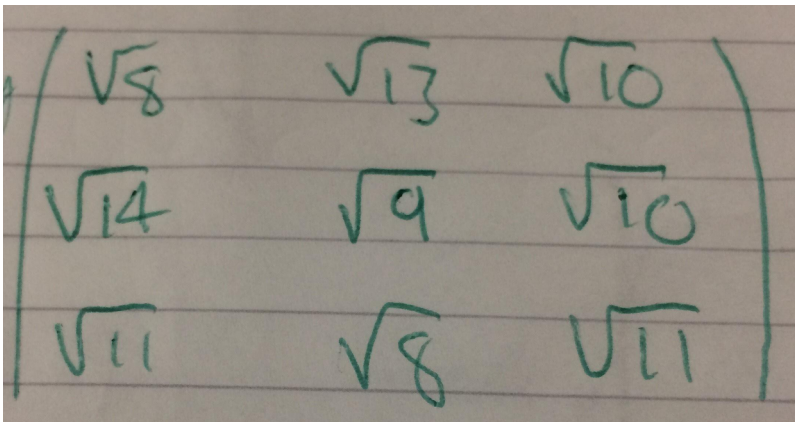
Testing:

Energy Calculation:

To test that the energy calculation was working correctly, I performed two tests. In the first I created a 3x3 matrix containing some very small RGB values (as this would be how Pixels are represented in code) on paper and hand calculated the energy of each element of the matrix using the equations given in the practical specifications. I only used small numbers so that this would be easier to calculate by hand. The input matrix of RGB values is shown below:



For this input, the required output, as specified by the equations in the specification, would be as follows:



The input fed into the program by code was the same as that shown on paper, and the corresponding output is shown below:

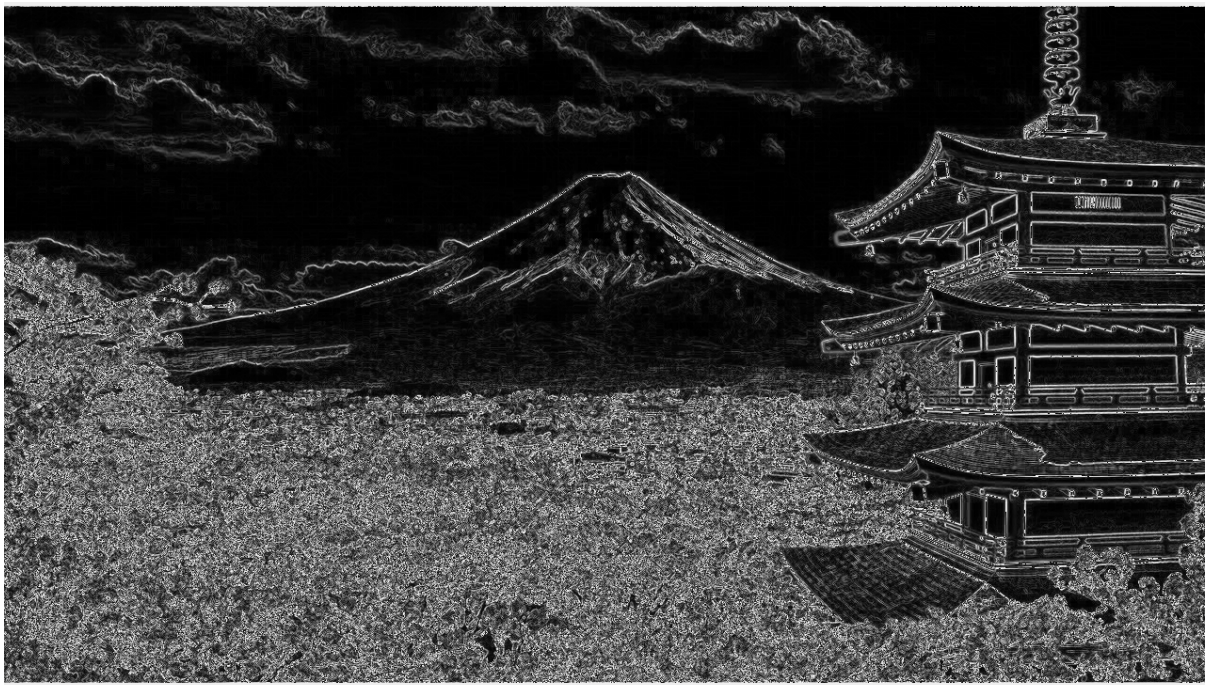
```
Line 1:
2.8284271247461903
3.605551275463989
3.1622776601683795

Line 2:
3.7416573867739413
3.0
3.1622776601683795

Line 3:
3.3166247903554
2.8284271247461903
3.3166247903554
```

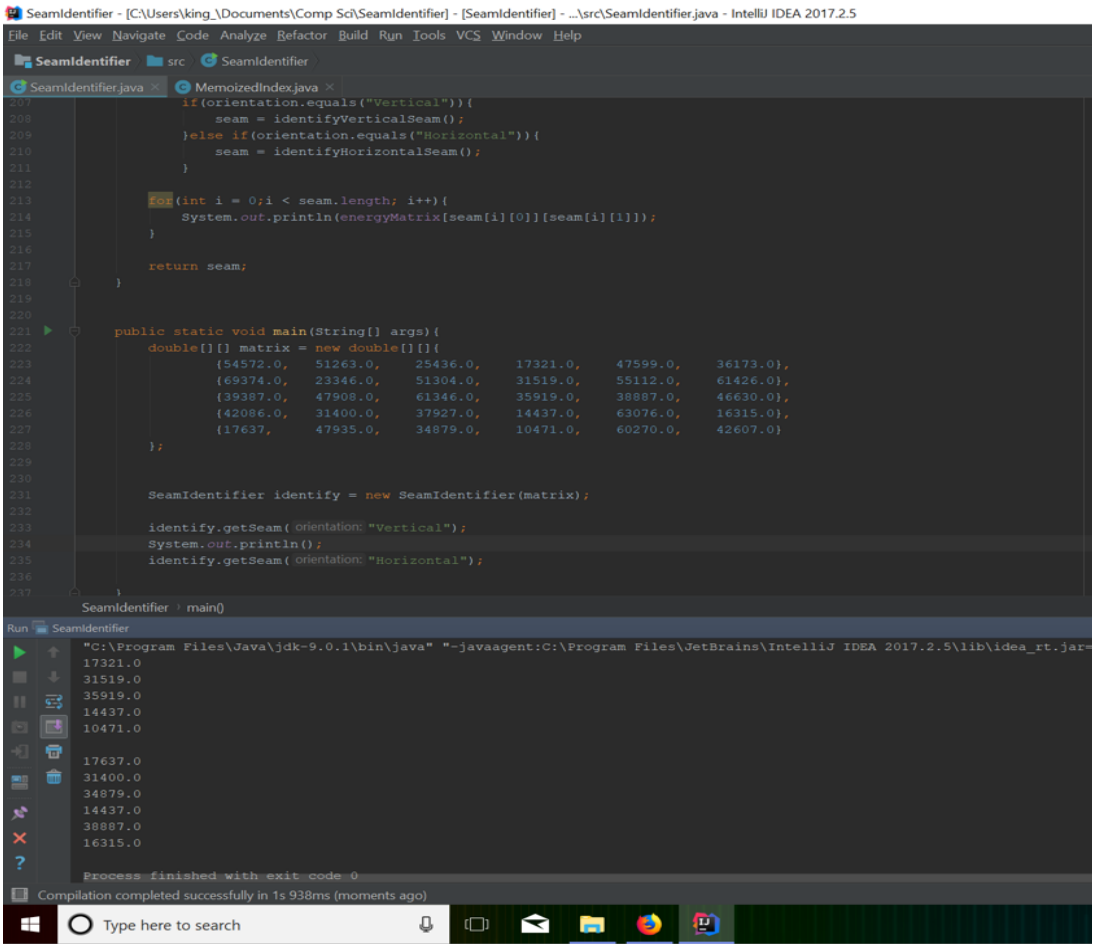
Calculating the values of the square roots gives us the values as given by the program and we can conclude that the energy

calculation algorithm is working correctly. To further test this, and to apply the energy calculation to images, I used the energy calculation module to output a greyscale image representing the energy of each pixel, where a brighter pixel has a higher energy. To do this, I set the R,G & B values of each pixel to the weight of the pixel. This produces a picture entirely in greyscale, and a test can be seen below. A successful output image should be dark in regions of single or similar colours and bright in regions with sudden or contrasting colour changes, and often the bright lines of the output will trace the outline of the subjects of the photo.



Seam Identification:

The seam identification was tested with a model matrix and then getting the horizontal and vertical seam from them and then outputting all the values in the seam.



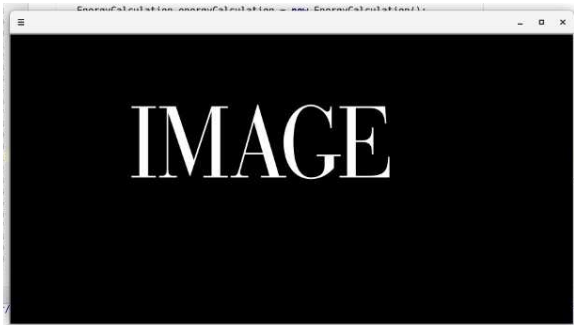
Program Function:

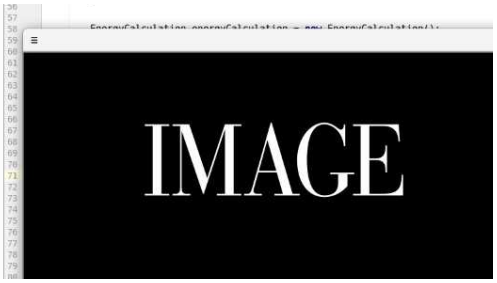
To test the overall program functionality we found a suitable image with large regions of low energy, similar colour pixels and some regions of high energy pixels and ran the program, selecting a size to crop the image to and compared the result to the original image. If we saw disruptions in the high energy areas we could have concluded that there were problems in the identification of the seam, but if the image looked similar to the original but smaller, we could conclude that the program was functioning normally. We used an image that was originally 1920x1280, and cropped it to 1800x1200 using the programs GUI. The image we used was titled ‘Purple Sunset’ and is shown below:

And after reducing its dimensions, it looks like this:

Crop Avoiding:

The crop extension will be tested by taking an image with lots of duplicate seams (black background) and then compare what the project specification solution would produce for that image against what the crop extension produces.





As can be seen on the left, although we forgot to use the same carve dimensions the image on the left (project specified result) has the word image tend towards the top left corner. The crop extension however produces a result where the important part of the image stay right in the middle.

Evaluation: Overall this program is very hard to evaluate without much scope for comparison. As is shown by the testing the classes all produce correct results and thus we can conclude that seam's are identified correctly. However some busier images often end up looking distorted if carved enough, this is only natural as eventually the lowest seam will be large, and as we remove seams the contrast between pixels often increases as colour gradients in the image fade. The program can also be critiqued for being very slow once high quality images are used, however this is a trade off we have made against having poorer carves. We considered not recalculating the energy after every seam removal but decided we would rather have more accurate seam removals. Another point would be that we could add another dimension of dynamic programming so that seams and energies that are unaffected by a seam removal don't need to be recalculated, however this would result in an overhaul of the entire program which we didn't have time for. The program however seems to work quite well and many images come out looking very good, however we are quite disappointed at how long some images take, it can be several minutes of waiting at times. In terms of the specification it does everything required and it is quite robust, so in that respect it is a good solution. The energies are calculated correctly, an individual seam identification uses dynamic programming (doesn't repeat calculations), and the seam removal removes a seam correctly. Another good aspect of our program is that the lowest seam from both vertical and horizontal orientations are compared to see which one should be removed when neither side has reached its destination length (doesn't just remove all the vertical seams and then all the horizontal ones). Finally the program also doesn't remove only the first or last lowest seam (assuming there are many duplicates) it alternatively removes the first and last so that we don't end up with an odd cropping effect.

Conclusion:

To conclude as very novice programmers we are very proud of what we achieved, we had many issues again with this project that took several hours to fix. As a result of this we didn't get to implement all of the extensions we had hoped to like saving an image, and showing where the seams on an image where removed as the seam carve went on. However we think we have

produced a program of a decent standard that although isn't by any means fully optimal it is completely functional.

ht44: I contributed much of the secondary function of the project in this case such as the handling of images and image data, the energy calculation, the seam carving and also the implementation of the Pixel class which we used throughout to complete the project. I also did much of the testing and writing the report.

Aj87: For this project I contributed the GUI as well as the seam identifier and the memoized index classes. I also helped work on the report and did a lot of empirical work.

Repo Link: https://ht44.hg.cs.st-andrews.ac.uk/seam-carving-repo/