CSci 433/502 Algorithms
Programming Assignment #2: Steganography
Spring 2016

Due: Monday, March 7th at midnight

Introduction

According to Wikipedia, steganography is "the practice of concealing a file, message, image, or video within another file, message, image, or video". Steganography is similar to cryptography since a message is encoded, but the difference is that the fact that a message has been embedded in the file, message, image or video is not obvious by simple inspection of the object. For this assignment, you will design and implement a program to embed a message into a file and retrieve a message from a file.

Image Format

Pixels in an image have 3 color components: red, green and blue. Commonly, each component is represented by an integer value in the range 0 to 255. When red=0, green=0 and blue=0, the pixel is black. The pixel is white when all three components have value 255.

One of the simplest file formats for storing an image is the PPM (Portable Pixel Map) file format. There are two different formats for PPM images: binary and ASCII. We will use the ASCII format, so that it is easily readable and writable. Unfortunately, the files can be very large when stored in this format.

The first line of an ASCII PPM file contains the value "P3" (no quotes). The second line of the file is a comment that begins with the # character. The third line of the file contains two integers: the width (number of COLS in the image) followed by the height (number of ROWS in the image). The fourth line of the file is a single integer that indicates the maximum color value for any component in the image. We will always use 255. So the header of an image file containing an image 200 pixels wide by 100 pixels high would look like:

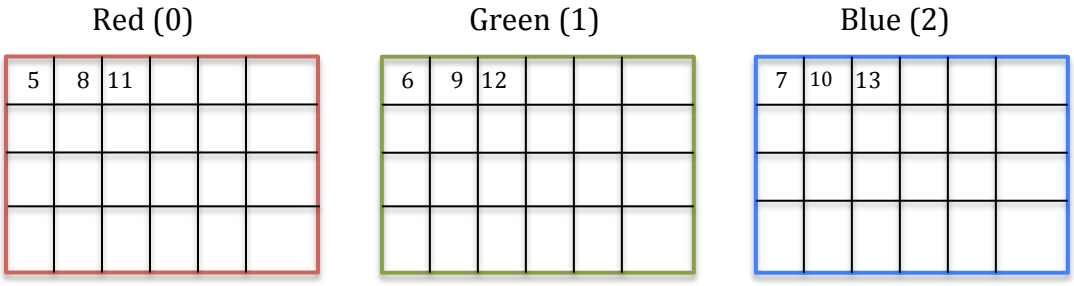
```
P3
# This is a comment describing the file
200 100
255
```

The fifth and following lines of the file contain component values. Lines 5, 6 and 7 have the values for the first pixel, red (R) then green (G) then blue (B). Lines 8-10 contain the RGB values for the second pixel, and so forth. The pixels are stored in row-major order (that is, across the first row, then across the second row, and so on).

PPM files can be viewed using many graphics programs, including gimp, which can be downloaded for free at gimp.org.

Requirements

You **MUST** include a class (object) in your program to store a PPM image. The actual representation of the image **MUST** include a three-dimensional array (rows by columns by color). The color dimension should be size 3 (values 0..2) for RGB (Red 0, Green 1, and Blue 2).



The number in the cell indicates the line number in the PPM that contains the value for the corresponding color component. So, image[0][0][0] is on line 5, and image[0][0][1] is on line 6, image[0][0][2] is on line 7, image[0][1][0] is on line 8, image[0][1][1] on line 9, image[0][1][2] on line 10, and so on.

* Create an object by passing the filename of the PPM file to the constructor.

* You must also be able to save an image to a file, in the same format as described above. Include your name in the comment one line 2 of every file created by your program. Test to be sure your program can read the file back in and construct an image correctly.

* You must create a hash function to give a semi-unique hash value for the image file. We will use the hash value as an indicator that you likely have embedded the message correctly in the image file. Here is the hash function that I devised for us all to use:

For each color dimension (RGB)
    Compute the sum of the even entries in that color plane
    Compute the sum of the odd entries in that color plane

Create a 6 character hash string (only uppercase letters) for the file by appending the following together in the order indicated:

1. H(RedEvenTotal)
2. H(RedOddTotal)
3. H(GreenEvenTotal)
4. H(GreenOddTotal)
5. H(BlueEvenTotal)
6. H(BlueOddTotal)

where H(*Total*) = (char) (Total % 26 + 65)

My sample output from reading in test file image1:

```
Reading in image from image1.ppm of size 194 by 259
Hash value is GMQUCL
```

* Your program must be able to extract a message from an image and to insert/embed a message into a PPM image file. You can assume that there is a properly encoded message embedded in the image files used for testing the extraction of a message. Here is the idea we will use of how to secretly store/retrieve a message in an image:

We will assume an embedded message ends with the special symbol '#'. To extract the message, consider the "sense" (odd or even) of each component of each pixel, in the order stored in the image file. An odd number implies a "1" bit and an even number indicates a "0" bit. If you pack all of the implied bits together, chunk them into 8 bit units, the message can be retrieved by converting each sequence of 8 bits to a character. When the '#' character is found, the message is complete. For example, if the first 8 values in message-enriched PPM file (beyond the header) are:
22
19
26
14
11
6
14
12

Then the bits would be 01010100, which in decimal is 84, which in ASCII is 'H'.

To encode a message, reverse the process to create a binary representation of the message (be sure to append a #). Then "adjust" the color components in the image file to match. Be sure to use 8 bits for each character (which may require

prepending some 0's). Recall that the color component values must be 0..255. So as you are processing the bits to encode, if the current bit is 0 and the color component is even or the current bit is 1 and the color component is odd, just output the same value for that color component. If the current bit is 0 and the color component is odd, then output the color component value-1 (the value was 1, 3, 5, …, 255, so the value -1 will still be in the range 0..254). If the current bit is 1 and the color component is even, output the color component value+1 (since the value was 0, 2, 4, …, 254, so the value+1 will be in the range 1..255).

Challenges

I will give out three prizes for this assignment. A student can claim at most one prize. Email me at dwilkins@cs.olemiss.edu to claim a prize. First come first served! I'll post on Blackboard when a prize is claimed.

First the prizes:

- 10 bonus points on a test
- A second dropped quiz
- A non-point related prize

The three challenges are:

1. Send me the correct hash values for files image2 and image3.
2. Send me the message encoded in image1M.
3. Encode the exact messages "Home Sweet Home" in image2, and the message "Beagles are the best dogs ever" in image3. Then send me the hash values of those two images with the embedded messages.

Submitting your work

Submit (1) a single zip (e.g. export) of your solution with embedded documentation and (2) the output from a *sample run of your program that fulfills all three challenges*. Upload to Program2 on Blackboard.