Hiding Messages in Plain Sight

Outline

Everything is Binary! 1

Under the hood, computers store everything as binary (all 0's and 1's).

Computers use various conventions to understand what the information really is. We will only go into the necessary details for dealing with images and text.

In base-10, $(1204)_{10}$ represents $1000+200+0+4$, or

$$1\times10^3+2\times10^2+0\times10^1+4\times10^0$$

Binary works the same way. $(1011)_2$ represents

$$1\times2^3+0\times2^2+1\times2^1+1\times2^0$$

A bit is a single digit of a binary number. A byte is eight bits. In $(1011)_2$, the last digit is the least significant bit.

How Computers Store Text 2

Computers use a convention called ASCII to convert between numbers and letters.

Letter	Base-10	Binary
A	65	01000001
B	66	01000010
D	?	?
a	97	?
–	?	01011111

QuickCheck: What is the least-significant bit of A?

So what is an image? 3

An image is made up of pixels, each of which is a color. The representation we give for pixels is three numbers:

- The "redness" of the pixel (a number between 0 and 255)
- The "greenness" of the pixel (a number between 0 and 255)
- The "blueness" of the pixel (a number between 0 and 255)

Putting these three numbers together, we call this the RGB value of the pixel.

QuickCheck: How many bytes is the R value of a pixel?

And? 4

Okay, so now, we know images are just a two-dimensional array of pixels. And each pixel is made up of three numbers between 0 and 255 (the RGB value). And each number is made up of binary digits. What can we do with this? Consider the following two sentences:

I am the color R=50, G=50, B=230.
I am the color R=51, G=51, B=231.

Can you tell the difference?

Humans are really bad at noticing differences in color

We can use the least-significant bits of each pixel to hide our message!

Steganography 5

Now we're ready to write pseudo-code for the main algorithm:

```
1 public PPMImage hidePhraseInImage(...) {
2    char[] textAsBits = convertStringToBits(...);
3    for each pixel in the image {
4       pixel[R] = setLastBit(pixel[R], /* next bit of text */);
5       pixel[G] = setLastBit(pixel[G], /* next bit of text */);
6       pixel[B] = setLastBit(pixel[B], /* next bit of text */);
7    }
8
9    //Question: What happens if we run out of message to hide?
10 }
```

This code uses two phantom functions `convertStringToBits` and `setLastBit`. Let's discuss these.

Converting a String to Bits 6

```
1 private char[] convertStringToBits(String s) {
2    char[] textAsBits = new char[s.length() * 8];
3    int index = 0;
4    for each character c in s {
5       for(int i=0; i < 8; i++) {
6          textAsBits[index] = getBit(c, i);
7          index++;
8       }
9    }
10 }
```

```
1 private char getBit(char c, int i) {
2    for (int j=0; j < i; j++) {
3       if (c is odd) { c -= 1; }
4       c /= 2;
5    }
6    if (c is odd) { return 1; }
7    else { return 0; }
8 }
```

Setting Bits 7

To write this function, we need to use bitwise operations. Luckily, these are similar to ones you've already seen!

Consider `p && q`. Assuming `p` and `q` are booleans, this returns true only when `p` and `q` are both true.

If `a` and `b` are bits, we can think of 0 as false and 1 as true. A bitwise-and looks like `a & b`, and returns 1 only when `a` and `b` are both 1. Similarly, `a | b` returns 0 only when `a` and `b` are both 0.

So, suppose we have a bit `b`, and we want to set it to 1. If we do `b | 1`, that will do the trick.

If we want to set `b` to 0, we can use `b & 254`.

Warning: In Java, bitwise operations return `int` types. You will want to cast to a `char`, like this:

```
(char) (b & 0)
```

PPM Images 8

The code you will write should support a type of image called PPM. The PPM format is as follows:

- The first line should be "`P3`" which is the type of PPM image format we're supporting.
- The second line should be "`h w`" where h is the height of the image and w is the width.
- The third line should "255" which represents the highest value of a color.
- All the remaining lines should be "`r g b`", where r, g, and b are the red, green, and blue values of the next pixel, in base-10.

Getting Text Out of the Image 9

We haven't discussed how to get the text back out of the image, but you should have everything you need to figure it out! Good luck! Have fun!

Methods to Implement 10

In `PPMImage`, you should implement:

- `public static PPMImage readPPM(String fileName)`
- `public boolean writePPM(String fileName)`

In `Steganography`, you should implement the helper functions:

- `private char setLastBit(char color, int value)`
- `private char getBit(char c, int i)`
- `private char[] convertStringToBits(String s)`
- `private String convertBitsToString(char[] bits)`

And, in `Steganography`, you should also implement:

- `public PPMImage hidePhraseInImage(PPMImage image, String textToHide)`
- `public String getPhraseFromImage(PPMImage image)`

You can grab the stubs at:

`http://countablethoughts.com/stego`