Extracting a Secret Text Hidden in an Image

Introduction on Steganography

Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. Generally, the hidden messages appear to be (or to be part of) something else: images, articles, shopping lists, or some other cover text.

In steganography, the intended secret message does not attract attention to itself as an object of scrutiny. This is in contrast to visible encrypted (concealed) messages. No matter how the concealed data could be revealed, they arouse interest.

Specification

You are provided with a .bmp image of a flower which looks like this:

A secret text is encoded in this image and your task is to extract it. The text is encoded as follows:

- In the original text to be encrypted, each character (for example, a letter) occupies one byte;
- Each byte is split into its individual bits;
- A .bmp image of a flower is then taken;
- Each bit of the text is then placed in the least significant bit of a byte in the picture file.

After this process, the image file looks very much as it did before. There is no sign whatever of the hidden text. Moreover, the number of bytes in the image file is exactly the same as it was before.

More specifically, this is how the encryption works:

- There is a header in the .bmp file which occupies 54 bytes. Nothing in this header is altered as this would corrupt the file.
- After that, we move forward *p* more bytes and this is where the first bit of the text is placed.
- Then, we move forward *q* bytes and place another bit of the text.
- We then continue moving forward *q* bytes, placing bits of the text until all the text is encoded.
- So, for example, suppose $p = 1$ and $q = 2$, then bits of the text will be in the bytes at positions $54+1$, $54+3$, $54+5$, $54+7$, etc.

To decrypt (reveal) the text, we simply reverse the steps. Given the values of *p* and *q* we:

- Start at position *54+p* and extract the least significant bit. This is the first bit of the encrypted text.
- Then go to position *54+p+q* and extract the bit.
- Then move forward *q* bytes at a time, extracting all the bits of text.
- Now we have all the bits. We now assemble them into bytes.
- We now have the text of the message which we write to a file.

So, given the values of p and q and knowing the algorithm as above, we can decrypt (recover) the text.

Note: You do not need to write the code to extract the text. It is provided for you!

Note: You do not need to understand the code of the decryption algorithm to do the assignment (though if you are interested in computers and programming it is well worth taking a look).

All you have to do is: Write a code that finds the correct values of p and q. The next section explains exactly how to do it.

How to Solve it

The starting point is `est_extract_secret_text.py`. You are provided with this program.

This extracts the hidden text from an image file using the provided values of *p* and *q*. However, we do not know the correct values of *p* and *q* to use. They are both set to 1 in the program you are given and these are **not** the correct values. **Your task is to alter this program so that it finds the required values.**

If you look in `est_extract()` you can see that *p* and *q* are simply set to 1. On the basis of the values chosen, function `est_extract_bits_from_image` is called and extracts the text. **So between the two, you must insert code to find the values of *p* and *q* and write out some simple lines of data (see below)**.

The way you are going to do it is as follows. **You need to write a function called `est_bit_proportion` which takes a Python bytearray as an argument and computes the proportion of ones in it**. You are going to apply this to the existing bytearray `message_byte_array` for a particular choice of values for *p* and *q*. **If the proportion is less than 0.5, then the chosen values for p and q are the correct ones to use in order to decode the text.** Only one combination of *p* and *q* values results in the proportion being less than 0.5.

At the point marked "# ADD CODE HERE" you are going to add code which will try different values of *p* and *q*, namely all combinations of *p = 1, 2, 3* and *q = 1, 2, 3*. There are thus **nine combinations** you are going to try. For each combination of *p* and *q*, you are going to run your function `est_bit_proportion` to find the proportion of ones. If the proportion of ones for some combination is less than 0.5, you have the answer. There is only one such combination in this assignment. Save those values of *p* and *q* and continue with the loop.

At the end, set the existing local variables *p* and *q* in `est_extract()` to the values you found in the loop. Then continue with the existing code to extract the hidden text and save it to a file.

Your program should run like this from the CMD command line (same as Assignment 1):

```
python cs_registrationnumber.py
```

The output of the program when run should be **exactly** in this format (only the actual values of *bp* and the values of *p* and *q* on the last line will be different):

```
p=1 q=1 bp=0.000
p=1 q=2 bp=0.000
p=1 q=3 bp=0.000
p=2 q=1 bp=0.000
p=2 q=2 bp=0.000
p=2 q=3 bp=0.000
p=3 q=1 bp=0.000
p=3 q=2 bp=0.000
p=3 q=3 bp=0.000
The answer is: p=1 q=1
```

The values for *bp* in your program **must be written to three decimal places in all cases, exactly as shown here**. 'bp' stands for 'bit proportion'. The actual values will all be different and will obviously not be 0.000. In fact they will all be fairly near to 0.500, e.g. a value might be something like 0.567 for example.

The result of running should be that the hidden text is extracted and written to a file called '`hidden_text_we_found.txt`' - the code for writing the file is already written for you. It already uses that filename.

If you look at '`hidden_text_we_found.txt`' having run your program, you will see it is the start of a famous novel. If '`hidden_text_we_found.txt`' contains random characters, your program has not worked! In other words, the values you found for *p* and *q* were not the correct ones.

Note: Everything must be written in Python 3 or above. **Please do not use Python 2.7 or any other programming language.**

Hints and Tips

1) **Remember, you only have to find values of *p* and *q*.** All the other code for extracting the hidden text from the image file is written for you. You only need to write about twenty simple lines of code (possibly less) in the place marked `# ADD CODE HERE` in the program, and to **write a simple function** called `est_bit_proportion` to compute the proportion of ones in a byte array.

2) To write `est_bit_proportion`, you can simply find the length of the bytearray using the built in function `len`, go round a loop counting all values in the bytearray which are 1, and then after the loop divide that count by (the length)*8.

For testing purposes, you can create a test bytearray like this (try typing it into python directly to experiment with this):

```
test = bytearray( 2 )
test[ 0 ] = 0b00000000
test[ 1 ] = 0b00000001
test
```

You can see that the contents of the bytearray are eight-bit bytes which we have specified, e.g., zero is written `0b00000000`. The program does some interesting things with byte operations but you do not need to follow the details or to write anything like that. All you need is to look at a particular byte in test and see if it is one or not. You can do that in the normal way like this:

```
test[ 1 ] == 1
```

Because Python knows that `test[ 0 ]` is a byte, it interprets 1 as meaning `0b00000001`. You could do that directly if you wanted to:

```
test[ 1 ] == 0b00000001
```

Now, continuing with our function, if we pass `test` as argument to `est_bit_proportion`, we will expect it to return 0.0625 as it is length 16 and one value is one while the other is zero (you must write the function first!):

```
est_bit_proportion( test )
```

3) You do not need to understand how the provided extraction code works in order to do the assignment. You can just treat it as a 'black box' if you like. However, for your interest, and to learn more about this encryption method, you might like to study it and to read the description of the algorithm which is given above. Also the code contains very detailed comments.

4) To write out something containing both strings and numbers in the same print statement using concatenation, you may need to convert numbers to strings using the built in function `str`, e.g.

```
x = 1
print( "x=" + x )
>>>ERROR
```

instead do

```
x = 1
print( "x=" + str( x ) )
```

Or alternatively, you can avoid the problem by doing

```
x = 1
print( "x=", x )
```

5) To write out any number to three decimals you can do:

```
"%.3f" % 1
"%.3f" % 1.3
"%.3f" % 1.34
"%.3f" % 1.345
"%.3f" % 1.3456
```

Notice that all these come out with exactly three decimals, as required. The last one is also rounded correctly.