PA7

PA7

CSE 8A Fall 2021 PA 7

Due date: Friday, December 3rd @ 11:59PM PST

(Late Submission is NOT Allowed. Redo will NOT be available for PA7.)

Provided Files

`CSE8AImage.py`, `steganography.py`, `M_M.bmp`, `CookieMonster.bmp`

Part 1: Implementation

In this PA, we will try to hide a message image within a context image. Then we will also implement a function to recover that message image from a combined image.
`CSE8AImage.py` provides image loading/saving and some other supports, do not change anything in it.
`steganography.py` contains starter code for this PA. Complete functions with `#TODO` comments by following the instructions below.

`def get_least_significant2(num)`

This method takes in an integer and returns the two least significant bits (rightmost two) as an integer.

- You can assume `num` will always be an integer between 0 and 255 inclusive.
- For example, `get_least_significant2(255)` should return 3 (since the rightmost two bits of 1111 1111 is 11) and `get_least_significant2(253)` should return 1 (since the rightmost two bits of 1111 1101 is 01).

`def get_most_significant2(num)`

This method takes in an integer and returns the two most significant bits (leftmost two) in decimal form.

- You can assume `num` will always be an integer between 0 and 255 inclusive.
- For example, `get_most_significant2(200)` should return 3 (since the leftmost two bits of 1100 1000 is 11) and `get_most_significant2(150)` should return 2 (since the leftmost two bits of 1001 0110 is 10).

`def embed_digits2(context_val, message_val)`

This method takes in two integers and returns the result of replacing the two least significant digits (rightmost two) in `context_val` with the value of `message_val`.

- You can assume that `context_val` will always be an integer between 0 and 255 inclusive (i.e. an 8-bit integer) and `message_val`will always be an integer between 0 and 3 inclusive (i.e. a 2-bit integer).
- Hint: One way to approach this problem is first to “clear” the two least significant digits of `context_val` (i.e., set them to 0) and then use addition to add in `message_val`. There are many ways you can clear these two digits, but a quick way is just to shift right and then shift left again. E.g.: 0011 1111 shift right 2 yields 0000 1111. If you shift 0000 1111 left 2 you will get 0011 1100.

`def hide_secret_message_2bits(context_img, message_img)`

This method takes in two 2D array of tuples (similar to the `image` format in PA6) and returns a new image with image `message_img` hidden in image `context_img`'s least significant 2 bits.

- You need to hide the most significant 2 bits of `message_img`'s RGB values in the least significant 2 bits of the corresponding RGB values of `context_img`.
- You can assume that `message_img` will always be the same size as `context_img`
- This method should make a copy of the image stored in `context_img` and modify and return that copy. It should not modify the input `context_img` image.
- Hint: make use of the functions that you already implemented above.

`def recover_secret_message_2bits(img_with_message)`

Given an image(which is a 2D array of tuples), this method should return a 2D array of tuples that is the hidden message image in `img_with_message`.

- You can assume that the message is hidden in the least significant 2 bits of the `img_with_message` image.
- Your method won’t know the dimension of the hidden image so all you need to do is to go through the entire `img_with_message` image. The returned image should have the same size as `img_with_message`.
- This method should not modify the input `img_with_message` image.
- Again, functions that you already implemented may help you.

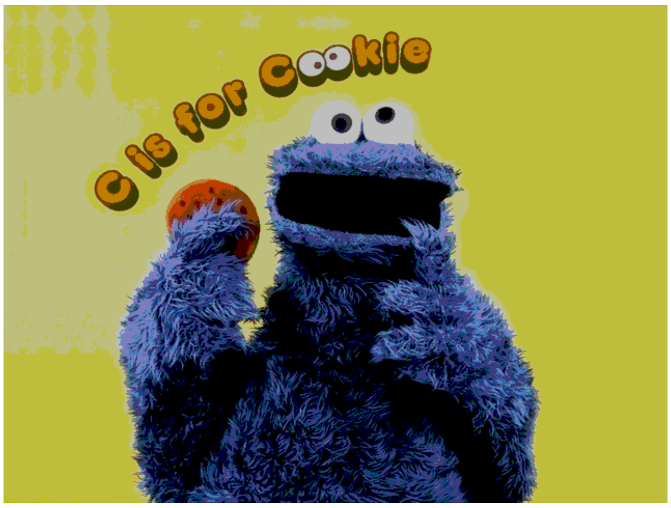
Hide and recover

After implementing all the above functions, use them to hide the CookieMonster image into the M_M image. In terminal, load `M_M.bmp` (the context image) and `CookieMonster.bmp` (the message image) as 2D arrays of tuples. Then call `hide_secret_message_2bits` to get the resulting image that encrypted the message image in the last two bits of the context image. Save the resulting image to a file called `M_MWithMessage.bmp`. Then call `recover_secret_message_2bits` on the resulting image and get the recovered image with message. Finally, save the recovered image to a file called `recMessage.bmp`.

- When finished, two additional files, `M_MWithMessage.bmp` and `recMessage.bmp` , should be created in your workspace.
- For your reference, the created images should look like below:

M_MWithMessage.bmp

recMessage.bmp



Extra credit for encoding using more or less than 2 bits (20 points)

If you want to do the extra credit part, create a file called `steganography_ec.py` and implement the following functions:

```
def get_least_significant_n(num, n_bits)
```

This method is a generalization of `get_least_significant2` in the mandatory part. It takes in an integer and returns the `n_bits` least significant bits (rightmost `n_bits`) in decimal form.

- You can assume `n_bits` is an integer between 1 and 7 inclusively.

`def get_most_significant_n(num, n_bits)`

This method is a generalization of `get_most_significant2` in the mandatory part. It takes in an integer and returns the `n_bits` most significant bits (leftmost `n_bits`) in decimal form.

- You can assume `n_bits` is an integer between 1 and 7 inclusively.

`def embed_digits_n(context_val, message_val, n_bits)`

This method is a generalization of `embed_digits2` in the mandatory part. Instead of replacing the two least significant digits (rightmost two) in `context_val` with the value of `message_val` , this function should return n least significant digits specified by input `n_bits`.

- You can assume `n_bits` is an integer between 1 and 7 inclusively.

`def hide_secret_message_nbits(context_img, message_img, n_bits)`

This method is a generalization of `hide_secret_message_2bits` in the mandatory part. The only difference is that image `message_img` should be hidden in image `context_img`'s least significant n bits specified by `n_bits`.

- You can assume `n_bits` is an integer between 1 and 7 inclusively.

`def recover_secret_message_nbits(img_with_message, n_bits)`

This method is a generalization of `recover_secret_message_2bits` in the mandatory part. The only difference is that image `message_img` is hidden in image `context_img`'s least significant n bits specified by `n_bits`.

- You can assume `n_bits` is an integer between 1 and 7 inclusively.

Part 2: Style

Coding style is an important part of ensuring readability and maintainability of your code. Starting from this assignment on, we will grade your coding style in all submitted code files (**including optional extra credit files if submitted**) according to the style guidelines. You can keep these guidelines in mind as you write your code or go back and fix your code at the end. For now, we will mainly be focusing on the following:

- **Use of descriptive variable names**: The names of your variables should describe the data

they hold. Your variable names should be words (or abbreviations), not single letters.
  - e.g. `a` --> `index_of_apple`; `letter1` and `letter2` --> `lower_case_letter` and `upper_case_letter`
  - Exception: If it is a loop index like `i`, `j`, `k`, one char is OK and sometimes preferred.
- **Inline Comments**: If there is a length of code that is left unexplained, take the time to type a non-redundant line summarizing this length of code (e.g. `#initialize an int` is redundant, vs. `#set initial length to 10 inches`). It will let others who look at your code understand what's going on without having to spend time understanding your logic first. But don't be too descriptive, as too many comments reduces readability.

Part 3: Conceptual Questions

You are required to complete the Conceptual Questions in PA lesson. There is no time limit but you must submit by the PA deadline. You can submit multiple times before the deadline. Your latest submission will be graded.

Submission

Turning in your code

Submit all of the following files to PA Lesson on EdStem via the "Mark" Button by **Friday, December 3rd @ 11:59PM PST. (Late submission is not allowed.)** Note that this is our last PA of this quarter, so due to the time constraint, we will not provide a Redo chance for this PA.

Evaluation

- **Correctness (75 points, +20 points for extra credit)**
  You will earn points based on the autograder tests that your code passes. If the autograder tests are not able to run (e.g., your code does not compile or it does not match the specifications in this writeup), you may not earn credit. You latest submission will be graded.
- **Style (5 points)**
- **Conceptual Questions (20 points)**

PA7 Conceptual Questions

Question 1 *Submitted Dec 1st 2021 at 8:54:56 pm*

Which of the following is the correct function to use to create a good picture for testing purposes?

- ○ `load_img`
- ○ `save_img`
- [✓ `create_img`]
- ○ `img_str_to_file`
- ○ None of the give answers is correct

Question 2 *Submitted Dec 1st 2021 at 8:57:17 pm*

We try to mirror an image from left to right around a vertical axis. What are the coordinates for the first three iterations of the inner loop? (assume picture has a height = 50 and width = 100)

```
img_w = width(img)
img_h = height(img)
mirrorPt = img_w // 2
for i in range(img_h):
    for j in range(mirrorPt):
        (rl, gl, bl) = img[i][j] #left
        img[i][img_w - j - 1] = (rl, gl, bl)#right
```

left	right
0, 99	99, 0
0, 98	98, 0
0, 97	97, 0

[✓]

left	right
0, 0	0, 99
0, 1	0, 98
0, 2	0, 97

left	right
0, 49	49, 0
0, 48	48, 0
0, 47	47, 0

left	right
0, 0	49, 0
1, 0	48, 0
2, 0	47, 0

None of the given choices is correct

Question 3 *Submitted Dec 1st 2021 at 8:59:18 pm*

Please pick the code that gives the column major copy of mirroring around a vertical axis.

[✓]

```
img_w = width(img)
img_h = height(img)
mirrorPt = img_w // 2
for j in range(mirrorPt):
    for i in range(img_h):
        (rl, gl, bl) = img[i][j]
        img[i][img_w - j - 1] = (rl, gl, bl)
```

```
img_w = width(img)
img_h = height(img)
mirrorPt = img_w // 2
for i in range(img_h):
    for j in range(mirrorPt):
        (rl, gl, bl) = img[i][j]
        img[i][img_w - j - 1] = (rl, gl, bl)
```

Question 4 *Submitted Dec 3rd 2021 at 7:26:49 pm*

What does this code do?

```
from CSE8AImage import *
def copy_region_to_new(img, r_s, c_s, r_t, c_t):
    img_w = width(img)
    img_h = height(img)
    new_img = create_img(img_h, img_w, (0,0,0))

    for row in range(100):
        for col in range(100):
            (r, g, b) = img[row+r_s][col+c_s]
            new_img[row+r_t][col+c_t] = (r, g, b)

    return new_img

pic = load_img('arch.jpg')
new_canvas = copy_region_to_new(pic, 10, 30, 50, 50)
save_img(new_canvas, 'result2.jpg')
save_img(pic, 'result3.jpg')
```

This code may generate an error

It copies the region but there is no way to access the picture with the copied region.

[✓ It copies the region and result2.jpg will show the effect]

- ○ It copies the region and result3.jpg will show the effect
- ○ None of the given answers is correct

Question 5 *Submitted Dec 1st 2021 at 9:00:35 pm*

How many total colors we can represent if each color channel uses 3 bits?

- ○ 3 * 3 * 3
- [✓ 8 * 8 * 8]
- ○ 256 * 256 * 256
- ○ 4 * 4 * 4
- ○ None of the given answers is correct