Laith Isbaitan 1190628

**PREPERATIONS:**

To save a message in the RGB LSB value of an image, we first consider that

1. Average length of a English word is 5 char  
   <https://www.wyliecomm.com/2021/11/whats-the-best-length-of-a-word-online/#:~:text=The%20average%20word%20in%20the,the%20Gunning-Mueller%20Fog%20Index>.
2. Average length of an English email is 100 words
3. Average number of bits in an English email = **100 word \* 5char \* 8bits = 4000bits**

With the previous info the receiving person can decipher and know how the message is spread-out without being given that info in the image.

1. We first calculate how many bits we can change in the LSB of the RGB of the pixels of an image, which is

**(height \* width of an image \* RGB = Height\*Width\*3)**

1. Then we divide the number of bits we can change In the image by the **average number of bits in an English email** **= imageBits/4000**, this would give us the number of space we can have between each bit of a secret message **(BIT\_SPACE)**, this would dilute(تحليل) the message in the image.

**HIDING MESSAGE:**

1. We change the string message into bits and pass it to the hideMessage method
2. We read the image and calculate the **BITS\_SPACE** and create the order of bits using shuffleOrder method
3. Then start looping through each pixel, and extracting the R G and B of that pixel. And pass that component to the modifyBit method
4. First we
5. modifyBit would first check if its time to put a bit from the messageToHide, if yes then reset the BIT\_SKIP and embed that bit in the LSB of the component
6. if its done with the message but still less than messageToHid.length()+8, then embed 1. This would make a stop sign.

**GET MESSAGE:**

1. Read the cover image and calculate the BITS\_SPACE and the ShuffleOrder
2. Loop through the pixels, and extracting RGB, then pass the component to extractBit method
3. extractBit would extract the LSB from RGB when the BIT\_SKIP = 0
4. after extracting the unordered bitMessage from the cover image, we reorganize it with reorganize() method.
5. Then we revert the bit message to a string message and print the charecters up to the stop sign which should be represented as eight 1’s (11111111) = ÿ.

This algorithm would hold messages up to the AVG\_BIT\_MESSAGE bit size, which is 4000 in my static base case.

And spread the bits of that message based on BIT\_SPACE.

Making 1 bit chunks, to dilute the message.

**SIGNATURE:**

To generate the signature of the message, we call the **getSignature** method that adds the message characters into a **Hashmap**, where we calculate the frequency of characters.

Then append the character and its frequency into a signature string. Ex:

**Message: “hi my name is laith”**

**Signature: ” 4a2s1t1e1h2i3y1l1m2n1”**