Homework 2

GWU CSCI 1112 - Fall 2019

September 18, 2019

1 Introduction

This homework is intended to reinforce your understanding of working with multi-dimensional arrays and to encourage you to think algorithmically. Your task is to develop an algorithm that solves the problem described using the provided framework.

1.1 Deadline

September 25, 2019 at 11:59pm

1.2 Submission

You must create a `.zip` file containing all of the files that you develop and work with and you must submit only the `.zip` file to blackboard before the deadline.

Your `.zip` file must be named using the following naming convention: `<yourNetId>-hw-02.zip`
For example, my NetId is `jrt`. If I submitted the homework, I would name my `.zip` file: `jrt-hw-02.zip`

1.3 Grading Rubric

- 50% For successful compilation
- 10% For sufficient comments
- 10% For consistent coding style
- 20% For base implementation
- 10% For extension implementation

1.4 Unit Testing

One significant change you will note from the previous homework is the introduction of a separate file, `StegUnitTest.java`, containing the individual unit tests and the `main` function. This architecture is more consistent with typical development and proper unit testing because the `Steganography` class can be used by any number of programs. This configuration also suggests how future assignments will be constructed and evaluated; however, you will not necessarily have access to the unit tests that you will be evaluated against and you will be expected to develop your unit testing to prove validation of your work.

The unit tests provided with this assignment show the basis for how this assignment will be automatically tested. We will also look at the implementations to ensure that you are not simply gaming the unit tests. Your implementation grade will be subject to the number of unit tests that your program passes.

1.5 Comments

I have provided relevant comments to document the files (a file header), the function signatures (function block comments), and inline comments in much of the `StegUnitTest.java` file. These comments are a model that you should follow in the future when developing files or functions on your own. This level of comments will be expected in future assignments.

You must substitute your name in the file header and provide any inline comments to document code that you contribute to your solution. You must also provide inline comments to the `testCopy` function in `StegUnitTest.java` to explain what each branch is testing for.

1.6 Plagarism

We will use a set of automated tools specifically designed to analyze code for plagarism. If you copy code from another source, classmate or website, there is a very high probability that these tools will flag your work as plagarized. You are permitted to discuss the problems at a high level; however, you must code your own solution. If you do not share code or outright borrow code from a website, you will have no problem with the plagarism filter.

2 Steganography

Steganography is the practice of encoding messages into and decoding messages from images. Steganography is a unique form of encryption because it is difficult to detect that a message has been encrypted into the image medium, so it can be used to encrypt in plain sight. The success of this practice is dependent on making minor changes that are difficult to detect so that these changes are indistinguishable from noise. For this exercise, we will use a very simple approach that could easily be improved for wider application.

Your implementation will copy the original key image into a new cypher image and will make subtle changes to the pixel values in the cypher image using a sparse pattern. Each pixel that you modify will be changed by an amount based on a the position of a letter in the alphabet. In order to spread out the error that we are introducing into the cypher image, **your implementation must change every tenth pixel in every tenth row beginning with the first row and first column**.

A framework for this program has been provided in `Steganography.java` and a set of unit tests have been provided in `StegUnitTest.java`. In order to compile and test your work, compile `StegUnitTest.java` and run `StegUnitTest`.

2.1 Base

The requirements in this section constitute a 90% solution. You must implement the following functions using the following function signatures:

```
public static int[][][] copy(int[][][] px)
public static int[] colorError( int[] keyPixel, int[] cypherPixel )
public static int[] positionToError( int chpos )
public static int errorToPosition( int[] error )
public static int[][][] encrypt(String s, int[][][] pxKey)
public static String decrypt(int[][][] pxCypher, int[][][] pxKey)
```

2.1.1 Base Functions

This section gives a general explanation of the requirements for each function. More information may be available in the function headers and in how the functions are unit tested.

```
int[][][] copy(int[][][] px)
```

`copy` performs a deep copy of the array of pixels `px` to another array and returns the deep copy.

```
int[] colorError( int[] keyPixel, int[] cypherPixel )
```

`colorError` computes the error in the color channels between two ARGB pixels. The array that is returned consists only of the error in the RGB channels.

```
int[] positionToError( int chpos )
```

`positionToError` computes the RGB color error from the position of a character in the alphabet. This function maps a character representation to color information. In the base requirements, the color error is uniform across all channels, so the same error is stored in every RBG channel. For example:

- 'A' is the first letter in the alphabet and in a zero based system it would be in position 0, so the RGB error would be [1,1,1].
- 'Z' is the last letter in the alphabet and in a zero based system it would be in position 25, so the RGB error would be [26,26,26].

```
int errorToPosition( int[] error )
```

`errorToPosition` computes the position of a character in the alphabet from the RGB color error. This function maps a color back to a character representation. By knowing the position of a character in the alphabet, we can then compute the ASCII representation of the character. In the base requirements, the color error is uniform across all channels, so the same error is stored in every RGB channel. For example:

- An RGB error of [1,1,1] indicates that the character encoded is the first character in the alphabet and in a zero based system this would translate to the value 0.
- An RGB error of [26,26,26] indicates that the character encoded is the last character in the alphabet and in a zero based system this would translate to the value 25.

```
int[][][] encrypt(String s, int[][][] pxKey)
```

`encrypt` takes a string consisting of only alphas (no numbers or non-alphabet characters), encodes the letters into a copy of the key image, and returns the newly encoded image. `encrypt` must use the functions `copy` and `positionToError`. `encrypt` only subtracts the error from the color channels in a pixel which is acceptable only because this is tested against a full color (white) image.

```
String decrypt(int[][][] pxCypher, int[][][] pxKey)
```

`decrypt` takes two images, a cypher containing an encoded message and a key image representing the original image that the message was encoded into, and returns a string consisting of only alphas (no numbers or non-alphabet characters). `decrypt` must use the functions `colorError` and `errorToPosition`.

2.2 Extension

The requirements in this section represents the remaining 10% of your potential grade. You will receive no credit for the extension if your base implementation is incorrect. Focus first on getting the base program fully correct, then consider the extension requirements. You must implement the following functions using the following function signatures:

```
public static int[] positionToError2( int chpos )
public static int errorToPosition2( int[] error )
public static int[][][] encrypt2(String s, int[][][] pxKey)
public static String decrypt2(int[][][] pxCypher, int[][][] pxKey)
```

2.2.1 Extension Functions

```
int[] positionToError2( int chpos )
```

`positionToError2` computes the RGB color error from the position of a character in the alphabet. This function maps a character representation to a color. In the extension requirements, the color error is spread across all channels, so only part of the overall error is stored in an individual color channel and the sum of the error across all channels encodes the character. For example:

- 'A' is the first letter in the alphabet and in a zero based system it would be in position 0, so the RGB error would be [1,0,0].
- 'B' is the second letter in the alphabet and in a zero based system it would be in position 1, so the RGB error would be [1,1,0].
- 'Z' is the last letter in the alphabet and in a zero based system it would be in position 25, so the RGB error would be [9,9,8].

This approach reduces the overall change in color for a pixel, so the changes are less obvious and a large range of values can be encoded into a pixel.

```
int errorToPosition2( int[] error )
```

`errorToPosition2` computes the position of a character in the alphabet from the RGB color error. This function maps a color back to a character representation. By knowing the position of a character in the alphabet, we can then compute the ASCII representation of the character. In the extension requirements, the color error is spread across all color channels, so only part of the error is stored in a single RGB channel. For example:

- An RGB error of [1,0,0] indicates that the character encoded is the first character in the alphabet and in a zero based system this would translate to the value 0.
- An RGB error of [1,1,0] indicates that the character encoded is the second character in the alphabet and in a zero based system this would translate to the value 1.
- An RGB error of [9,9,8] indicates that the character encoded is the 26th character in the alphabet and in a zero based system this would translate to the value 25.

```
int[][][] encrypt2(String s, int[][][] pxKey)
```

`encrypt2` takes a string consisting of only alphas (no numbers or non-alphabet characters), encodes the letters into a copy of the key image, and returns the newly encoded image. `encrypt` must use the functions `copy` and `positionToError2`. `encrypt2` will be compared against a gradient where some channels are no color and some channels are full color. When encoding the error, if the color channel is closer to no color than full color add the error for that channel otherwise subtract the error for that channel.

```
String decrypt2(int[][][] pxCypher, int[][][] pxKey)
```

`decrypt2` takes two images, a cypher containing an encoded message and a key image representing the original image that the message was encoded into, and returns a string consisting of only alphas (no numbers or non-alphabet characters). `decrypt` must use the functions `colorError` and `errorToPosition2`.