Zachary Roth
CS1800
4/20/21

Final Project: Hexadecimal Text Visualizer

Introduction

For my project, I decided to create a program using Python that can take a .txt file as input, and produce a visualization of the hexadecimal digits found within the text file. This is done by creating and putting together pixels with RGB values equal to the hexadecimal digits found in the text file. In addition, the decimal values of the hexadecimal digits are also sorted in ascending and descending order by insertion and selection sort algorithms, respectively, and printed as grayscale images. To manage complexity, improve portability, and make the code easier to read, I used a multitude of functions to perform the operations needed to create an image.

Analysis

From a coding perspective, the program starts by importing the math module to perform complex math operations, the Pillow module to create images, and numpy module for array-based operations. A list of the valid hexadecimal digits is also declared, with digits 0-9 and uppercase and lowercase letters from A to F. Additionally, a brightness scale variable was added as well, as most images come out very bright, which will be used in a later function. This is because most text files, like a movie script or book, contain massively more letters than numbers, and letters have higher values than numbers when it comes to hexadecimal, which will in turn make brighter RGB colors.

The first function, *get_hex_list()*, loops through each character in a given file. If a given character is a member of the valid hexadecimal list, then that character is added to the list of hex characters.

The next function, *hex_to_decimal_list(hex_list)*, takes in a list of hexadecimals as an argument. Using a make-shift switch case, it converts each hexadecimal in the list to a decimal value. This will help with sorting the list and also is needed for Pillow's functionality.

The next function, *selection_sort_descending(num_list)*, performs a selection sort of descending numbers for a list of numbers, in this context, the list of decimals. It uses two for loops, one to increase the current comparing variable, and another to iterate through the rest of the unsorted list to find a new maximum in the list. The result is a sorted, descending list.

The next function, *insertion_sort_ascending(num_list)*, performs an insertion sort of ascending numbers for a list of numbers, also a list of decimals in this context. It uses a for loop to loop through each element in the list, and a while loop to place the element in their appropriate place in the list.

The next function, *decimal_to_eightbit_list*(*decimal_list)* creates an "8 bit" list from the decimals it is handed to. Basically, it takes every two decimals, and treats them as two linked hexadecimals, which represent a total of eight bits. It then converts the two decimals to another decimal value, this time from 0-255 instead of 0-15. For example, if given a list with 6 and 13, the output will be 109, as 6*16 + 13 = 109. This is done for every two decimals in the list, and to make sure that the input list length is even, it removes the last number if the length is odd to make an even number for the calculations. This function was created so the decimals can comply with a future step that will need RGB data for every pixel that will

be displayed in the image. Typically, RGB data is stored in 24 bits, with 8 bits for each color of a pixel, red, green, and blue. Sometimes RGB values can be displayed with a string of six hexadecimal digits, which is where I got the inspiration from to make this project.

The next function, *darken_eightbits(eightbit_list)* takes in an eightbit list. Essentially, because the top 6 values for hexadecimal digits are letters, and most text files have a majority of letters, the image comes out very bright. This is due to the higher RGB values, as higher values mean brighter pixels, with 255,255,255 being completely white. To help the image come out a little less bright, a brightness constant, that is a float from 0 to 1, can be changed to bring down the values for each 8-bit, as a multiplier (or divider in this case).

The last function, *create_pixel_array(eightbit_list)* takes an eightbit list and groups the eight bit list into groups of three eight-bits, or 24 bits, which is used to represent an RGB value, and in turn, a pixel. These 24-bit values are then grouped into a square array. This square array represents the image, with each array element being a pixel. The majority of 24 bit list lengths will not have a whole square root, so the next lowest whole square root was calculated, and extra 24 bit RGB values were omitted from the final image.

Lastly, the program comes together by asking the user what the name of their .txt file is. It must be in the same folder as the program to work. The program takes some time to compute the images for the given file. With larger files, like books and movie scripts, the process can take quite some time, especially with the sorting algorithms. This is due to the fact that the insertion and selection algorithms are O(n^2) on average, which means that it is inefficient and costly with resources when it comes to large amounts of data. The program will then produce four images, the unsorted, selection-sorted, insertion-sorted, and darkened visualizations. The program is then complete.

Results

To develop and test my program, I used the script of the classic 2007 family/comedy hit, Bee Movie. After running the program, four images were produced, shown below.

Figure 1: Bee Movie: Unsorted Visualization

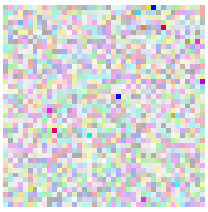


Figure 2: Bee Movie: Selection-Sorted (Descending) Visualization

Figure 3: Bee Movie: Insertion-Sorted (Ascending) Visualization

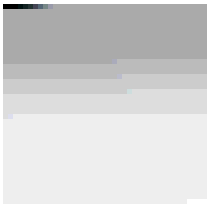
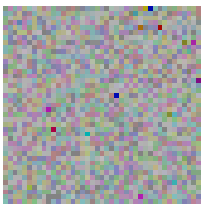


Figure 4: Bee Movie: Darkened Unsorted Visualization



The first and fourth images seem to be random pixels, but each pixel represents the first six, second six, and so on hexadecimals of the text file, from left to right, and then to the next line down. The first picture seems to have lots of lighter pixels, as the hexadecimal digits that are letters represent higher, hence brighter, values. The second and third pictures are in grayscale. This is due to the selection and insertion sort, and how it was done on the list of the decimal values. This list of decimal values after the sort looks like [0,0,0… 15,15,15], which will become [(0, 0, 0) … (255, 255, 255)]. The lower values are represented as darker colors, and the higher as brighter. The Pillow module builds images from arrays going to the right and then down, which can be seen as the colors get brighter/darker depending on the sort as you progress to the right and down the pixels. The insertion sort is ascending, so the pixels get brighter as you traverse through. The selection sort is descending, so the pixels get darker as you traverse through the image. Also, the resolution will go up depending on how large the text file is. A long poem might be a 4x4 image. An entire encyclopedia could go up to an HD image. Even with movie scripts, the pixels are still clearly visible in these above images.

Conclusion

In the end, multiple concepts that have been taught throughout the CS1800. More specifically, number representation and algorithms. Number representation was used by converting hexadecimal values to decimal values, and also computing decimal values of two-digit hexadecimal numbers (*decimal_to_eightbit_list*). Sorting algorithms were used to sort the decimal lists, in ascending order with the insertion sort, and descending order with selection sort. The program also showcases how Big O works in the real world, as the two sorting algorithms are O(n^2), which takes up a lot of time and resources with large amounts of data. If the user puts in a small text file, the process will be relatively short, within seconds. However, if the user inputs a book, like I tried with the 1984 novel, it can take several minutes to sort through. I thoroughly enjoyed this course and making this project, as I was able to showcase some of the information I learned in this course, in a form that I enjoy, which is programming. This project also helped me become more familiar with Python, as I am still learning about the language.

To run this project, make sure that the filename of the .txt you give it is in the same folder as the .exe. The images will also be saved to this folder.