CS111 Intro to Computer Science Fall 2017

Milestones Project

Iteration # 1

Leonardo Roman lr534

1. Count how many red pixels there are in an image

As it is described, the pixels can be represented as 2-D array. If we wish to count the number of different type of elements, as in this case color objects, it is needed to traverse the entire array to determine the count of each type of element. For example, let's say we have a n by n 2-d array filled up with red and blue boxes and we wish to count the number of red pixels in the array.

R	B	R	B
B	R	B	R
R	B	R	B
B	R	B	R

The following algorithm would count only the red pixels inside the 2-D matrix:

```
for( i = 0; i < number of rows; i++)
for(j = 0; j < number of columns; j++)
if (Arr [i][j] == red)
increment counter by one
```

In the pseudocode above the nested for loop will iterate every single element in the array by checking every element row by row. An integer variable "counter" will be needed to be declared and initialized to zero in order to store the number of red pixels. This can be achieved, by incrementing counter every time a red pixel is found. Red pixels are found, as the program traverse through the array and by comparing the object's property, in this case color properties, with the red color.

2. Change all the red pixels in an image to blue pixels and vice versa

Eg:

For this assignment, the order of the matrix can be manipulated by implementing an insertion algorithm. Such algorithm will need to store the value of any particular targeted index ,temporarily, to later be swapped with any other index value. In this case if we need to swapped positions of red with the one with blue, the algorithm would be described as follow:

```
for( i = 0; i < number of rows; i++)
for(j = 0; j < number of columns; j++)
if (Arr [i][j] == red AND Arr [i][j+1] == blue)
temp = red
Swap red with blue
Swap blue with temp
```

Since we are dealing with objects and object's properties, the variable temp will need to be an instance of the same object in order to be able to store any data of that same object, in this case pixels.

3. Take the left half of an image and reflect it over an imaginary vertical line going across the middle of the image.

Eg:

For assignment number 3 we can use almost the same steps used in assignment 2 with the only difference that this time we will be overwriting the right half of the image with the inverse pixel of the left side of the image.The algorithm would be as followed:

```
declare new array and assigned the inverse of left half of image to it
for( i = matrix number of rows; i > 0 ; i--)
for( j = matrix number of columns / 2; j > 0; j--)
newArr[i][j] = Arr[i][j]
populate right half of image with newArr
for( i = 0; i < number of rows; i ++)
for( j = number of columns / 2; j < number of columns; j++)
Arr[i][j] = newArr[i][j]
```

The first nested for loop assigns all element in the left half of the image such that new array contains the inverse image of the left side of the image. While the second nested for loop will populate the inverse to the left side of the image. This is accomplished by iterating up to half of the columns and traversing backwards. And by filling up the right side starting by half of the size of the columns up to the max number of columns.

4. Take the top half of an image and reflect it over an imaginary horizontal line

going through the image

Eg:

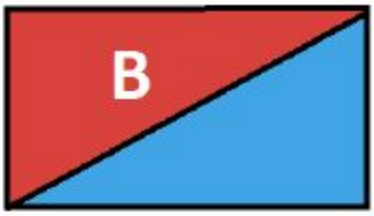
In this case we can use almost the same steps used in assignment 3 with the only difference that this time we will be overwriting the bottom half of the image with the inverse pixel of the top side of the image.The algorithm would be as followed:

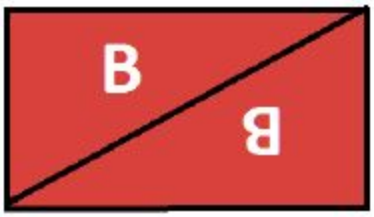
```
declare new array and assigned the inverse of top half of image to it
for( i = matrix number of rows / 2; i > 0 ; i--)
for( j = matrix number of columns; j > 0; j--)
newArr[i][j] = Arr[i][j]
populate bottom half of image with newArr
for( i = number of rows  / 2; i < number of rows; i ++)
for( j = 0; j < number of columns; j++)
Arr[i][j] = newArr[i]j]
```

The first nested for loop assigns all elements from the top of the image to new array such that new array is the reverse image of the the top side of the image. While the second nested for loop will populate with the inverse of the top to the bottom of the image. This is accomplished by iterating up to half the size of rows in array and traversing backwards. And by filling up the bottom part of the array starting from half of the size of the rows up to the max number of rows.

5. Take the upper half of an image and reflect it over an imaginary diagonal line going from the upper left corner of the image to the lower right corner.

Eg:





For this part we need an algorithm that takes that divides the array by a diagonal. If we can make an algorithm such that keeps track of all pair elements, $\{a_{03}, a_{12}, a_{21}.....a_{ij}\}$ for $\sum_{j=colum}^{0} \sum_{i=0}^{row} a_{ij}$ the reflection of the above picture can be achieved. Example:

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

To draw the diagonal a nested for loop would be used such that the image is divided by this line. The algorithm will need to store all elements from row to the limit "the line" and then swap them with the elements on the other side of the line. A pseudo code would look like as follow:

```
for (i =number of rows - 1; i > 0; i--)
for (j = 0, row= i; row <= number of columns - 1; j++, row++)
save the values temporarily
for (i = 0; i <= number of rows - 1; i++)
for (j = 0, y = i; y <= number of columns - 1; j++, y++)
swap
```