Compressing the Gray Value Data Set for a Gray Image

Anmol Singh Sethi IIT2016040 | Anirudh Singh Rathore IIT2016091 | Dheeraj Chouhan IIT2016078 | Manish Ranjan IIT2016059 | Vikas Kumar IIT2016058

***Abstract*—In this report a strategy to compress the given data set which is a matrix representing the gray values of a gray image has been developed.**

***Index Terms*—gray value, gray-scale image, matrix, compression**

I. Introduction and Literature Survey

An image consists of a two-dimensional array of numbers. The color or gray shade displayed for a given picture element (pixel) depends on the number stored in the array for that pixel. The simplest type of image data is black and white. It is a binary image since each pixel is either 0 or 1.
The next, more complex type of image data is gray scale, where each pixel takes on a value between zero and the number of gray scales or gray levels that the scanner can record. These images appear like common black-and white photographs they are black, white, and shades of gray. Most gray scale images today have 256 shades of gray.

For example consider the following matrix:

$$\begin{pmatrix} 35 & 1 & 6 & 26 & 19 & 24 \\ 3 & 32 & 7 & 21 & 23 & 25 \\ 31 & 9 & 2 & 22 & 27 & 20 \\ 8 & 28 & 33 & 17 & 10 & 15 \\ 30 & 5 & 34 & 12 & 14 & 16 \\ 4 & 36 & 29 & 13 & 18 & 11 \end{pmatrix}$$

The matrix given above represents an image, conversion of this matrix into an image yields the following image:-

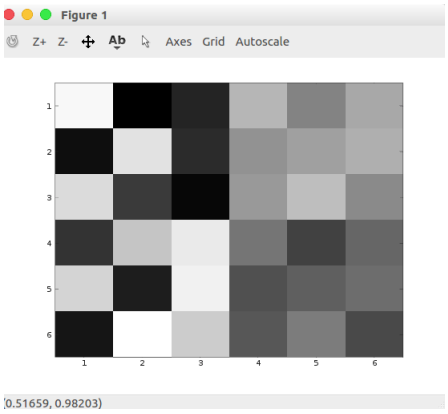

Fig. 1. Image representation of matrix

The problem statement required us to convert the image's gray value data set into another matrix by averaging the cells whose gray values had difference less than or equal to one.

II. Algorithm Design And Explanation

We adopted a simple algorithm in order to solve the problem. Once we have obtained the matrix for the gray scale image we do the following steps:-
1. Start from the first cell of the matrix, check in all the possible directions (here, right and down) in clockwise manner and update the value of the adjacent cells whose values differ by not more than 1. In general elements to be checked are the ones present at top, top-right diagonal, right, right-bottom-right diagonal, bottom, bottom-left diagonal, left, up-left diagonal elements.
2. After completing one cycle for the first cell, move on to the second cell and continue till the last element of the matrix.
3. Exit the program,
The pseudo-code for the algorithms described above is the following:-
Input: An NxM matrix which represents the gray scale values of a gray image.
Output: Compressed matrix which represents another image.

Consider the following:-

$$\begin{pmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{pmatrix}$$

The matrix given above represents an image, conversion of this matrix into an image yields the following image:-

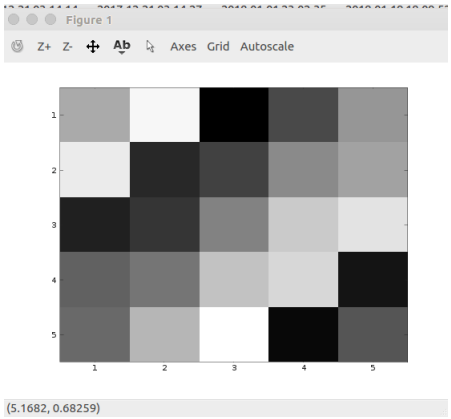

Fig. 2. Before

After passing the it's gray value matrix through the algorithm, we get the following values :

$$\begin{pmatrix} 17.00 & 23.50 & 1.00 & 7.50 & 15.50 \\ 23.50 & 5.50 & 7.50 & 13.50 & 15.50 \\ 4.00 & 5.50 & 13.50 & 20.50 & 22.00 \\ 10.50 & 12.00 & 18.50 & 20.50 & 2.50 \\ 10.50 & 18.50 & 25.00 & 2.50 & 9.00 \end{pmatrix}$$

On converting the above matrix into it's respective image, we get the following:

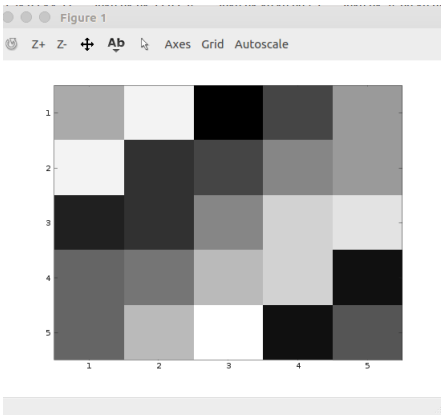

Fig. 3. After

We can clearly see that some parts of the image are blurred and some of the adjacent cells in the matrix representation of the gray image have same gray values. The image is not of not the same quality.

III. Time Complexity Analysis and Discussion

For the problem the time complexity is $\theta(n*m)$. Intuitively speaking, we can say that for the complete traversal of the matrix takes time of order *n* x *m*. And since there are N Rows and M columns in the matrix the time-complexity will be *n* x *m*. But consider the following proof for the mathematical analysis:-

Algorithm 1 compress data from given grayscale image

```
procedure Check(x,y)
    if (x>y && (x-y)<= 1) then
        return1
    if (x<y && (y-x)<= 1) then
        return1
    return0

procedure Compress()
    for ( i = 0 to n-1) do
        for ( j = 0 to m-1) do
            if (i>=0 && Check(array[i-1][j] , array[i][j]) =
1) then
                new = (array[i][j] + array[i-1][j])
                new = new / 2.0
                array[i][j] = array[i-1][j] = new
            if (i>=1 && j+1>=m && Check(array[i-
1][j+1],array[i][j]) = 1) then
                new = (array[i][j] + array[i-1][j+1])
                new = new / 2.0
                array[i][j] = array[i-1][j+1] = new
            if (j+1<m && Check(array[i][j+1],array[i][j]) =
1) then
                new = (array[i][j] + array[i][j+1])
                new = new / 2.0
                array[i][j] = array[i][j+1] = new
            if (i+1<n && j+1<m && Check(array[i+1][j+1]
, array[i][j]) = 1) then
                new = (array[i][j] + array[i+1][j+1])
                new = new / 2.0
                array[i][j] = array[i+1][j+1] = new
            if (i+1<n && Check(array[i+1][j],array[i][j]) =
1)) then
                new = (array[i][j] + array[i+1][j])
                new = new / 2.0
                array[i][j] = array[i+1][j] = new
            if (j-1>=0 && i+1<n&& Check(array[i+1][j-
1],array[i][j]) = 1)) then
                new = (array[i][j] + array[i+1][j-1])
                new = new / 2.0
                array[i][j] = array[i+1][j-1] = new
            if (j-1>=0&& Check(array[i][j-1],array[i] [j]) =
1)) then
                new = (array[i][j] + array[i][j-1])
                new = new / 2.0
                array[i][j] = array[i][j-1] = new
            if (i-1>=0 && j-1>=0&& Check(array[i-1][j-
1] , array[i][j]) = 1)) then
                new = (array[i][j] + array[i-1][j-1])
                new = new / 2.0
                array[i][j] = array[i-1][j-1] = new

procedure main
    Compress();
    for ( i = 0 to n-1) do
        for ( j = 0 to m-1) do
            print(array[i][j])
```

The time units required for the complete traversal for average case is as follows:-
time = 3m + 3n + 188 mn + 4
mn <3m + 3n + 188 mn + 4 <198mn
g(x) <f(x) <h(x)
g(x) is the tightest lower bound of the function and h(x) is the least upper bound of the function. The time complexity is of the order m*n. For the sake of simplicity and to avoid confusion, consider m equal to n. Thus the time complexity is of the order n^2.
1. *Worst Case* : Consider an NxM matrix filled with the same gray value in all the cells. For such a case, the output will remain the same but every time the computation will take place and values will be updated. Thus more time is taken. This the time taken is of the order n^2 which is the worst-case. This concludes our worst-case scenario. For e.g.

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

2. *Best Case*:- Consider an NxM matrix filled with numbers such that any adjacent element has a number which differ by more than one. In such a case only the values will be checked and no value will be updated. For e.g.

$$\begin{pmatrix} 17 & 19 & 21 \\ 27 & 29 & 31 \\ 33 & 35 & 37 \end{pmatrix}$$

The time complexity will be of the order n^2.
3. *Average Case*:- Consider an NxM matrix that is filled with numbers that are randomly generated. The time take to compute this will lie between the best-case and the worst-case. It is difficult to compute the time for such a case in general because it will be purely case-dependent. But the limits can be found.

To verify the theoretical claim, we plotted the graph between T VS N, to find out that the worst-case if proportional to $198n^2$ and the best-case is proportional to n^2 and all other cases lie in this range.

WORST CASE = O(n^2)
BEST CASE = $\Omega(n^2)$
AVERAGE CASE = $\Theta(n^2)$

IV. Experimental Study

In this section of the repost the actual data of the time calculation has been shown. The time plots have been constructed using the unit time equations at every step of the computation. Further the data was written in a file and GNU plots were constructed to verify our claims in the previous sections. All the values have been obtained considering unit time.

TABLE I
Time Complexity Table- Part(b)

n	**t_{best}**	**$t_{average}$**	**t_{worst}**
100	50930418	51383340	80023218
200	405050968	408686352	638756568
300	1364361518	1376651440	2154199918
400	3230862068	3260052708	5104353268
500	6306552618	6363563350	9967216618
600	10893433168	10992049508	17220789968
700	17293503718	17450168824	27343073318
800	25808764268	26042666348	40812066668
900	36741214818	37074350338	58105770018
1000	50392855368	50849906212	79702183368

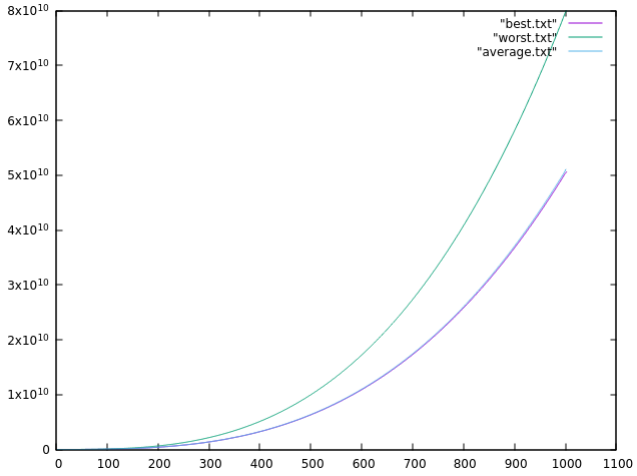

Fig. 4. Part(b) T(y-axis) VS N(x-axis)

The average and best case nearly overlap each other while worst case can be seen evidently.

V. Conclusion

Here, we have tried to design an efficient algorithm compresses the data-set which represents an gray image.

References

[1] http://introcomputing.org/image-6-grayscale.html/

[2] https://in.mathworks.com/company/newsletters/articles/how-matlab-represents-pixel-colors.html

[3] http://homepages.inf.ed.ac.uk/rbf/BOOKS/PHILLIPS/cips2ed.pdf