Assignment No : C3

1 Aim :

The aim of this assignment is to generate a Huffman code from the given gray scale image.

2 Problem Statement :

Generate Human codes for a gray scale 8 bit image.

3 Learning Objectives :

1. To understand the basics of image compression and Huffman Codes generation

4 Learning Outcomes :

1. After successfully completing this assignment, you should be able to understand Implement generation of Huffman code for a given gray scale imge using openCV and eclipse IDE.

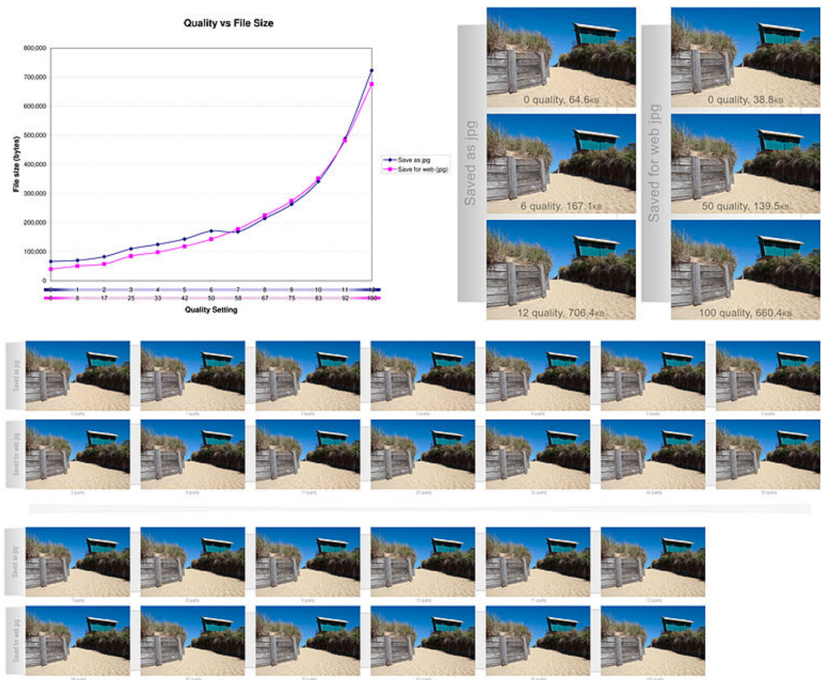
5 Software and Hardware Requirement :

1. 64-bit Fedora or equivalent OS with 64-bit Intel i5/i7 or latest higher processor computers.
2. Latest Open source update of Eclipse Programming frame work, GTK+ openCV version 2.4.13

6 Concept Related Theory:

6.1 Image Compression:

Image compression may be lossy or lossless. Lossless compression is preferred for archival purposes and often for medical imaging, technical drawings, clip art, or comics. Lossy compression methods, especially when used at low bit rates, introduce compression artifacts. Lossy methods are especially suitable for natural images such as photographs in applications where minor (sometimes imperceptible) loss of fidelity is acceptable to achieve a substantial reduction in bit rate. The lossy compression that producible differences may be called visually lossless.



Region of interest coding. Certain parts of the image are encoded with higher quality than others. This may be combined with scalability (encode these parts first, others later) Meta information. Compressed data may contain information about the image which may be used to categorize, search, or browse images. Such information may include color and texture statistics, small preview images, and author or copyright information. Processing power. Compression algorithms require different amounts of processing power to encode and decode. Some high compression algorithms require high processing power. The quality of a compression method often is measured by the Peak signal-to-noise ratio. It measures the amount of noise introduced through a lossy compression of the image, however, the subjective judgment of the viewer also is regarded as an important measure, perhaps, being the most important measure.

6.2 Huffman Encoding

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding and/or using such a code proceeds by means of Huffman coding, an algorithm developed by David A. Huffman while he was a Ph.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".[1] The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (weight) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted.[2] However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods.

7 Mathematical Model:

Let S be the system under consideration, S={s, e, X, Y, fme, DD, NDD, Su, Fa}

where, Identify the inputs S =Start state

E=End state

Fn= fread ,fdisplay ,fhuffman

X= x —x [0—255] N

Y = HFcodes

DD = Deterministic data;

NDD = Non deterministic data; here the image format other than TIFF format is invalid

1. Operations performed :

fread ,fdisplay ,fhuffmani

fread = reads the gray scale image

fdisplay = display gray scale image specifying its dimensions(height, width) and pixel values

fhuffman = it generates the Huffman code for the given gray scale image

2. Output:

O = Huffman code of the given gray scale image.

8 Program Code:

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <stdlib.h>
#include <iostream>
#include <opencv2/imgproc/imgproc.hpp>
#include <math.h>
#include <queue>
```

```
#include <vector>
#include <iomanip>

using namespace std;

struct node
{
        int c;
        int f;
        struct node *left;
        struct node *right;
};

class compare
{
        public:
    bool operator()(node *l, node *r)
        {
                return (*l).f>(*r).f;
        }
};

node* getNode()
{
        node *temp = new node;
        temp->c = '\0';
        temp->f = 0;
        temp->left = NULL;
        temp->right = NULL;
        return temp;
}

void printCodes(node *head, string str)
{
        if(head == NULL)
                return;
        if(head->left == NULL && head->right == NULL)
        {
                cout<<head->c<<"\t\t"<<str<<endl;
                return;
        }
        if(head->left)
                printCodes(head->left, str+"0");
        if(head->right)
                printCodes(head->right, str+"1");
        return;
```

```
}

void HuffmanCodes(int arr[], int freq[], int size)
{
        priority_queue<node*, vector<node*>, compare> minHeap;
        node *head = NULL, *ptr1 = NULL, *ptr2 = NULL;
        for(int i = 0; i < size; i++)
        {
                node *temp = getNode();
                temp->c = arr[i];
                temp->f = freq[i];
                minHeap.push(temp);
        }
        while(minHeap.size() > 1)
        {
                ptr1 = minHeap.top();
                minHeap.pop();
                ptr2 = minHeap.top();
                minHeap.pop();
                node *temp = getNode();
                temp->f = ptr1->f + ptr2->f;
                temp->left = ptr1;
                temp->right = ptr2;
                minHeap.push(temp);
        }
        head = minHeap.top();
        minHeap.pop();
        string str;
        cout<<"Alphabet\tCode\n";
        printCodes(head, str);
        return;
}

int main()
{
        cv::Mat src = cv::imread("1.png", CV_LOAD_IMAGE_GRAYSCALE);
        if(!src.data)
        {
            cout<<"Error opening file.\n";
            return -1;
        }
        int wgt[256], freq[256], arr[256];
        cv::namedWindow("Source Image", CV_WINDOW_AUTOSIZE);
        cv::imshow("Source Image", src);

        for(int i = 0; i < 256; i++)
```

```
                wgt[i] = 0;
        for(int i = 0; i < src.rows; i++)
                for(int j = 0; j < src.cols; j++)
                    wgt[src.at<uchar>(i, j)]++;

        int j = 0;
        for(int i = 0; i < 256; i++)
                if(wgt[i] != 0)
                {
                        arr[j] = i;
                        freq[j] = wgt[i];
                        j++;
                }

    HuffmanCodes(arr, freq, j);
    cv::waitKey(0);
    return 0;
}
```

9 Output:

```
Inspiron-5547:~/cl1$ cd c3left
Inspiron-5547:~/cl1/c3left$ g++ c3.cpp `pkg-config opencv --cflags --libs`
Inspiron-5547:~/cl1/c3left$ ./a.out
Alphabet        Code
76              0000000
51              0000001
78              0000010
245             000001100000
247             000001100001000
250             000001100001001
251             000001100001010
252             000001100001011
246             0000011000011
242             00000110001
240             0000011001
6               000001101
19              00000111
88              0000100
77              0000101
212             00001100
28              00001101
72              0000111
79              0001000
```

13	00010010
20	00010011
81	0001010
85	0001011
91	0001100
86	0001101
73	0001110
87	0001111
49	0010000
206	00100010
180	00100011
84	0010010
29	00100110
205	00100111
52	0010100
207	00101010
208	00101011
236	001011000
237	001011001
21	00101101
74	0010111
50	0011000
68	0011001
90	0011010
102	00110110
27	00110111
112	00111000
22	00111001
83	0011101
70	0011110
89	0011111
235	010000000
7	010000001
25	01000001
203	01000010
26	01000011
54	0100010
202	01000110
106	01000111
71	0100100
67	0100101
60	0100110
105	01001110
24	01001111
59	0101000
82	0101001

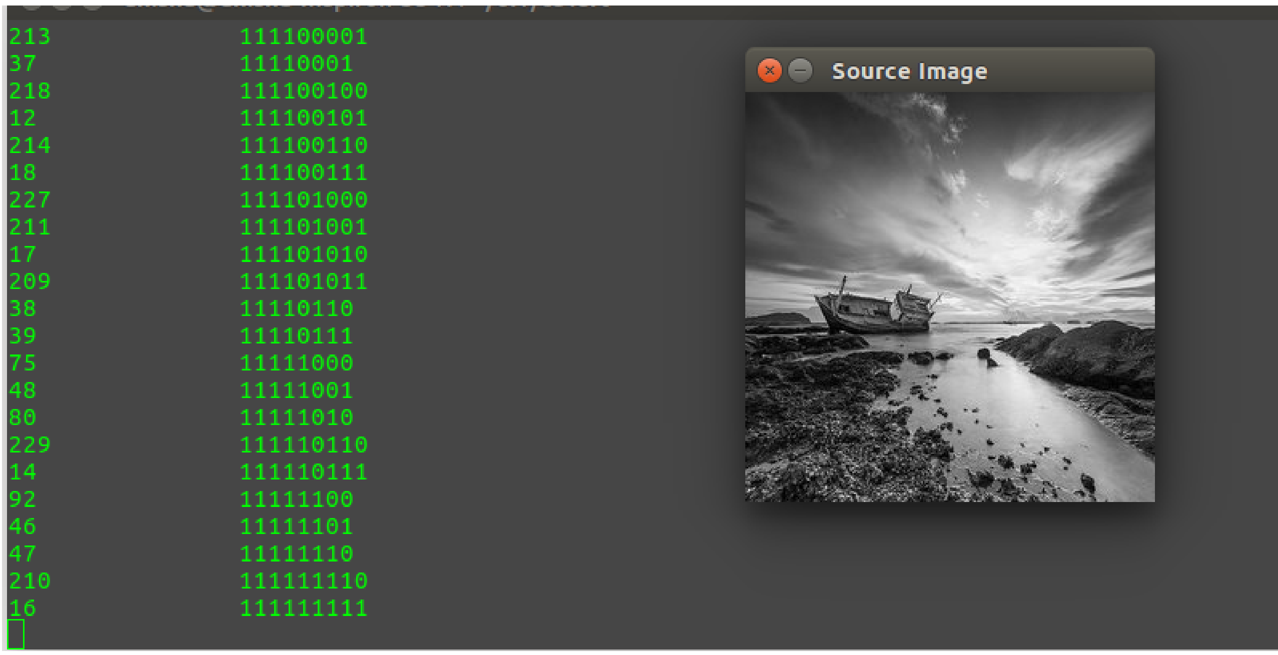
204	01010100
33	01010101
62	0101011
57	0101100
23	01011010
179	01011011
66	0101110
56	0101111
108	01100000
177	01100001
58	0110001
186	01100100
181	01100101
176	01100110
31	01100111
55	0110100
69	0110101
103	01101100
241	0110110100
2	0110110101
239	0110110110
244	011011011100
243	011011011101
1	01101101111
144	01101110
201	01101111
65	0111000
107	01110010
99	01110011
53	0111010
118	01110110
115	01110111
63	0111100
119	01111010
30	01111011
32	01111100
109	01111101
64	0111111
172	10000000
200	10000001
183	10000010
169	10000011
147	10000100
104	10000101
101	10000110
173	10000111

149	10001000
114	10001001
116	10001010
174	10001011
168	10001100
117	10001101
189	10001110
185	10001111
223	100100000
9	100100001
113	10010001
167	10010010
171	10010011
111	10010100
143	10010101
61	1001011
140	10011000
165	10011001
145	10011010
195	10011011
197	10011100
192	10011101
151	10011110
166	10011111
141	10100000
155	10100001
142	10100010
124	10100011
110	10100100
184	10100101
139	10100110
121	10100111
234	101010000
224	101010001
199	10101001
188	10101010
178	10101011
175	10101100
233	101011010
0	101011011
150	10101110
182	10101111
157	10110000
123	10110001
193	10110010
187	10110011

148	10110100
134	10110101
190	10110110
3	1011011100
238	1011011101
225	101101111
100	10111000
136	10111001
198	10111010
191	10111011
146	10111100
154	10111101
122	10111110
221	101111110
222	101111111
95	11000000
42	11000001
138	11000010
196	11000011
97	11000100
162	11000101
131	11000110
10	110001110
232	110001111
194	11001000
158	11001001
170	11001010
153	11001011
164	11001100
152	11001101
98	11001110
130	11001111
160	11010000
126	11010001
96	11010010
120	11010011
159	11010100
137	11010101
125	11010110
161	11010111
35	11011000
156	11011001
34	11011010
135	11011011
132	11011100
8	110111010

226	110111011
163	11011110
133	11011111
40	11100000
93	11100001
127	11100010
94	11100011
228	111001000
230	111001001
44	11100101
43	11100110
216	111001110
231	111001111
41	11101000
129	11101001
220	111010100
11	111010101
45	11101011
128	11101100
217	111011010
4	1110110110
5	1110110111
36	11101110
15	111011110
219	111011111
215	111100000
213	111100001
37	11110001
218	111100100
12	111100101
214	111100110
18	111100111
227	111101000
211	111101001
17	111101010
209	111101011
38	11110110
39	11110111
75	11111000
48	11111001
80	11111010
229	111110110
14	111110111
92	11111100
46	11111101
47	11111110

```
210                 111111110
16                  111111111
Inspiron -5547:~/cl1/c3left$
```



10 Conclusion:

Thus, we have successfully implemented Huffman Encoding for an 8-bit grayscale image.