Compression/Decompression of Images Using Huffman Coding

Alaa Allah Essam Mera
ID: 9191781
alaa.mera99@eng-st.cu.edu.eg

Aya Mahmoud Salam Okasha
ID: 9180361
aya.okasha99@eng-st.cu.edu.eg

Mariam Ashraf Mohammed
ID: 9181380
mariam.aziz99@eng-st.cu.edu.eg

Meeran Ahmed Mostafa
ID: 9181482
Meeran.elsayed99@eng-st.cu.edu.eg

Nouran Khaled Soliman
ID: 9191777
nouran.youssef99@eng-st.cu.edu.eg

I. INTRODUCTION

Nowadays , it is needed to deal with small files in size. Some files are big size so we need to compress them without losing any of its properties and information.Huffman encoding is an algorism that compress any file to be smaller in size (it's the basic concept of file compression)

Its principle is using variable-length encoding instead of fixed-length encoding of character (0,1) depending on the frequency of each character in the text to control the size.

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.[1] However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods - it is replaced with arithmetic coding or asymmetric numeral systems if better compression ratio is required.

Huffman coding is such a widespread method for creating prefix codes that the term "Huffman code" is widely used as a synonym for "prefix code" even when such a code is not produced by Huffman's algorithm.

II. MOTIVATION

We have chosen this project for some reasons

- After studying the idea of project , We made sure to apply the parts that are studied in this semester like struct ,tree ,queue, map…etc and use them into Huffman algorithm.
- Inquisitiveness and new knowledge about image process.
- Inquisitiveness of compression as a whole.
- We took it as a challenge compared our knowledge and we wanted to step deeper into data structure.

III. LIBRARIES

- Libraries

Iostream , String library , vector library , cstdint , queue , map , cstddef,and bitset.

- Tools
  - Qt creator
  - Vs code

IV. CHALLENGES AND PROBLEMS

The following problems where the challenges that faced for us

- Image reading : It took us a while to reach a simple method to read the picture through streams Especially ifstream , and all the methods we found was dealing with text files while we wanted to deal with them as they are (binary files) without the need to convert them to text files.(solved)
- After Huffman tree implementation , we spent a long time searching for a method to serialize the image until we found ifstream library and it was hard to deal with it.(solved)
- Padding problem.(solved)
- We had a big proplem with the decoding, the function logic is correct but at the same time we had a segmentation fault and when we debugged the program we found errors like (wild pointers , alloc (bad memory location) , malloc , and munmap_chunk: invalid pointer) they're all memory access errors we don't know how to deal with them.(solved)

- GUI , dealing with its classes is new to us . making relation between our implemented functions and actions in gui.(solved)

V. User Manual for the System

./compress "image name"

./decompress "encoded file name"

The technique works by creating a binary tree of nodes. These can be stored in a regular array, the size of which depends on the number of symbols. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the symbol itself, frequency of appearance of the symbol and optionally a link to a parent node which makes it easy to read the code (in reverse) starting from a leaf node. Internal nodes contain a frequency, links to two child nodes and an optional link to a parent node. As a common convention, bit '0' represents following the left child and bit '1' represents following the right child. A finished tree has up to n leaf nodes and n-1 internal nodes. A Huffman tree that omits unused symbols produces the most optimal code lengths.

The simplest construction algorithm uses a priority queue where the node with lowest probability is given highest priority:

1. Create a leaf node for each symbol and add it to the priority queue.
2. While there is more than one node in the queue:
   a) Remove the two nodes of highest priority (lowest probability) from the queue
   b) Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
   c) Add the new node to the queue.

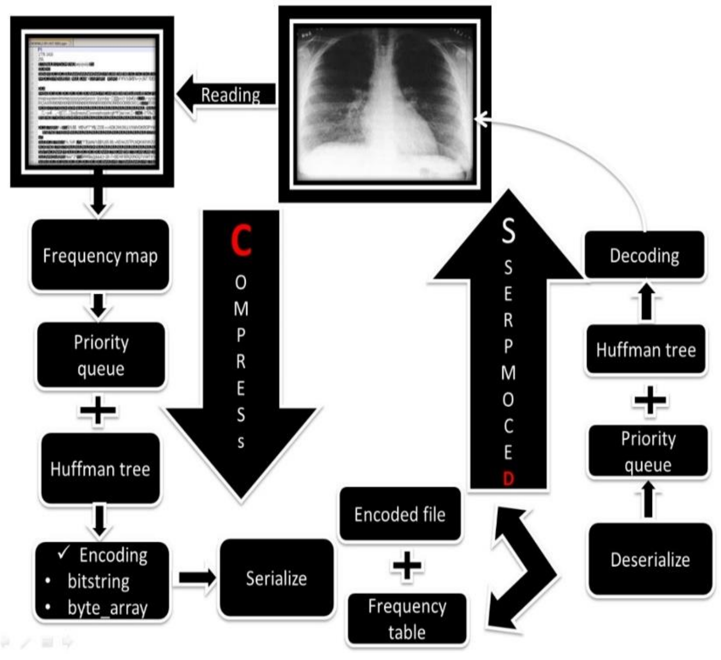


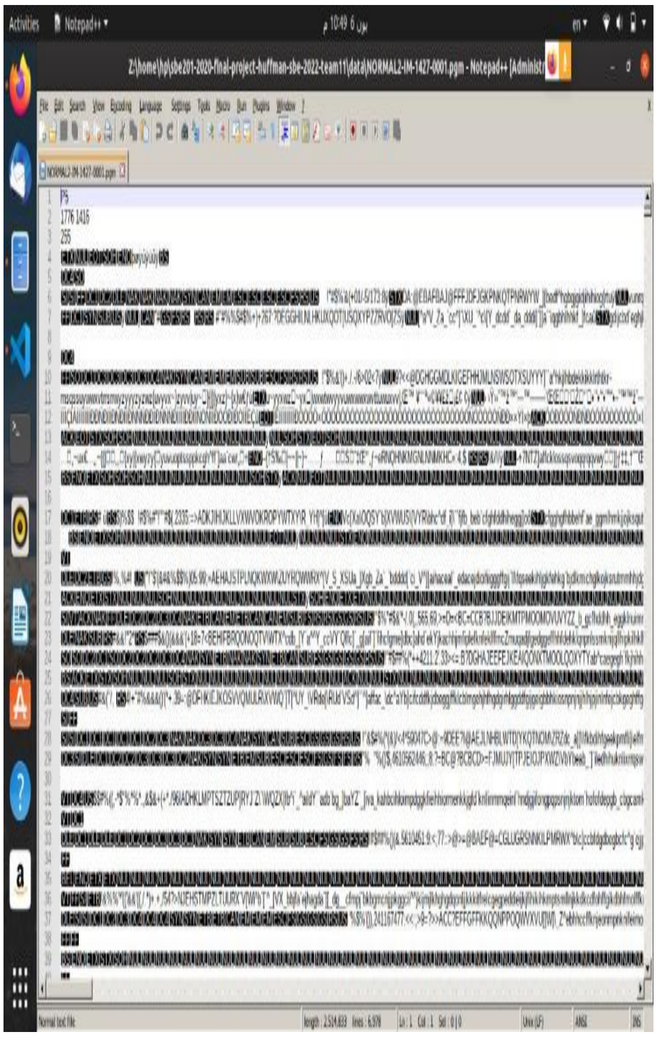
Fig 1: Block diagram of the system

3. The remaining node is the root node and the tree is complete.

VI. Contributors

- Reading : Nouran and Alaa.
- Frequency map:Mariam ,Meeran and Nouran.
- Huffman tree and encoding : Mariam and Meeran.
- Decoding : Mariam, Aya and Meeran.
- Serialization & compress: Nouran and Alaa
- Decompress: Alaa
- Deserialization : Nouran.
- Improvement: Mariam and Aya.
- Report : Aya and Meeran.
- GUI : Alaa , Meeran and Mariam

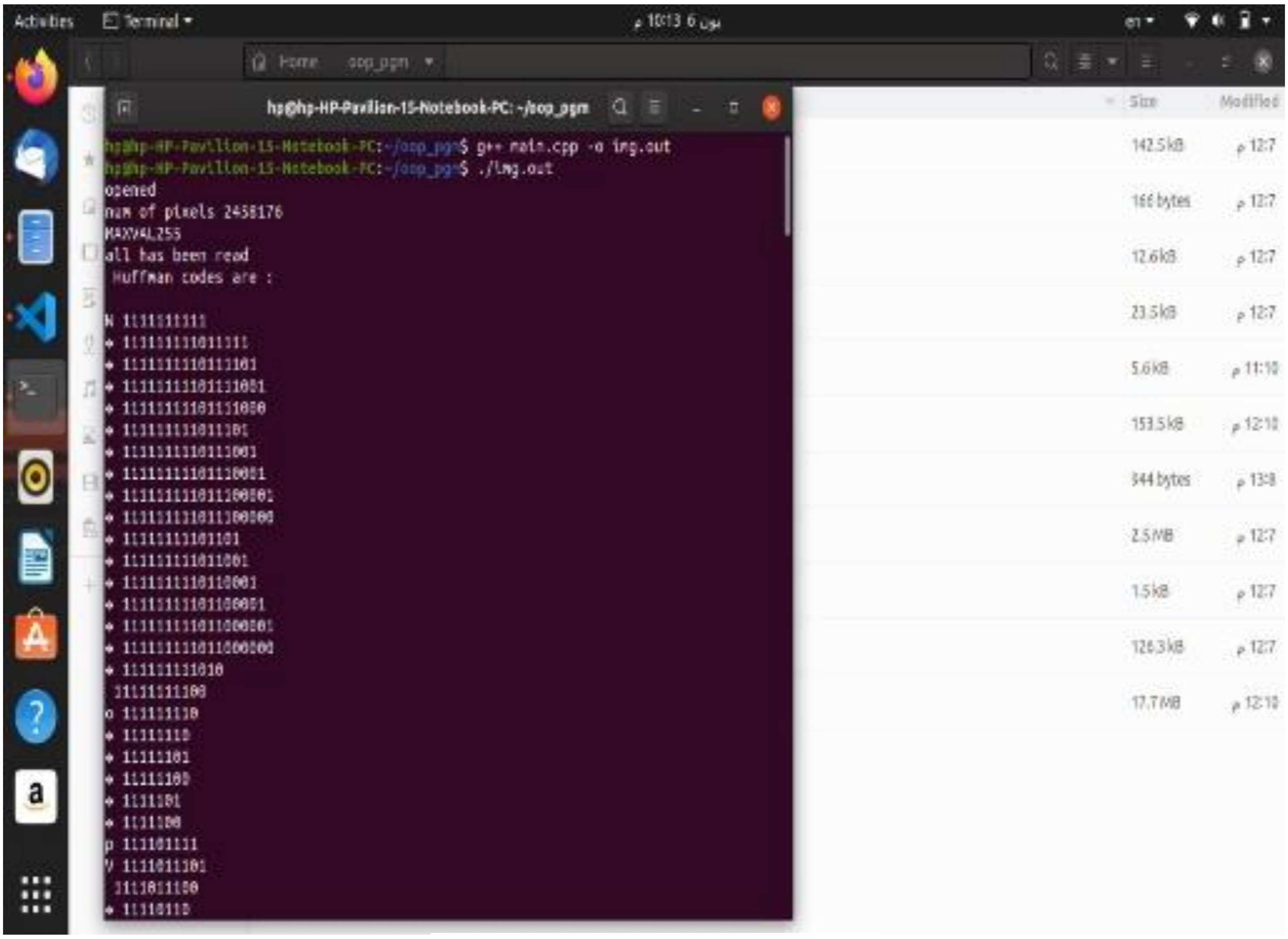
VII. Results


Activities
Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
NORMAL2-IM-1427-0001.pgm

Z:\home\hp\sbe201-2020-final-project-huffman-sbe-2022-team11\data\NORMAL2-IM-1427-0001.pgm - Notepad++ [Administrator]

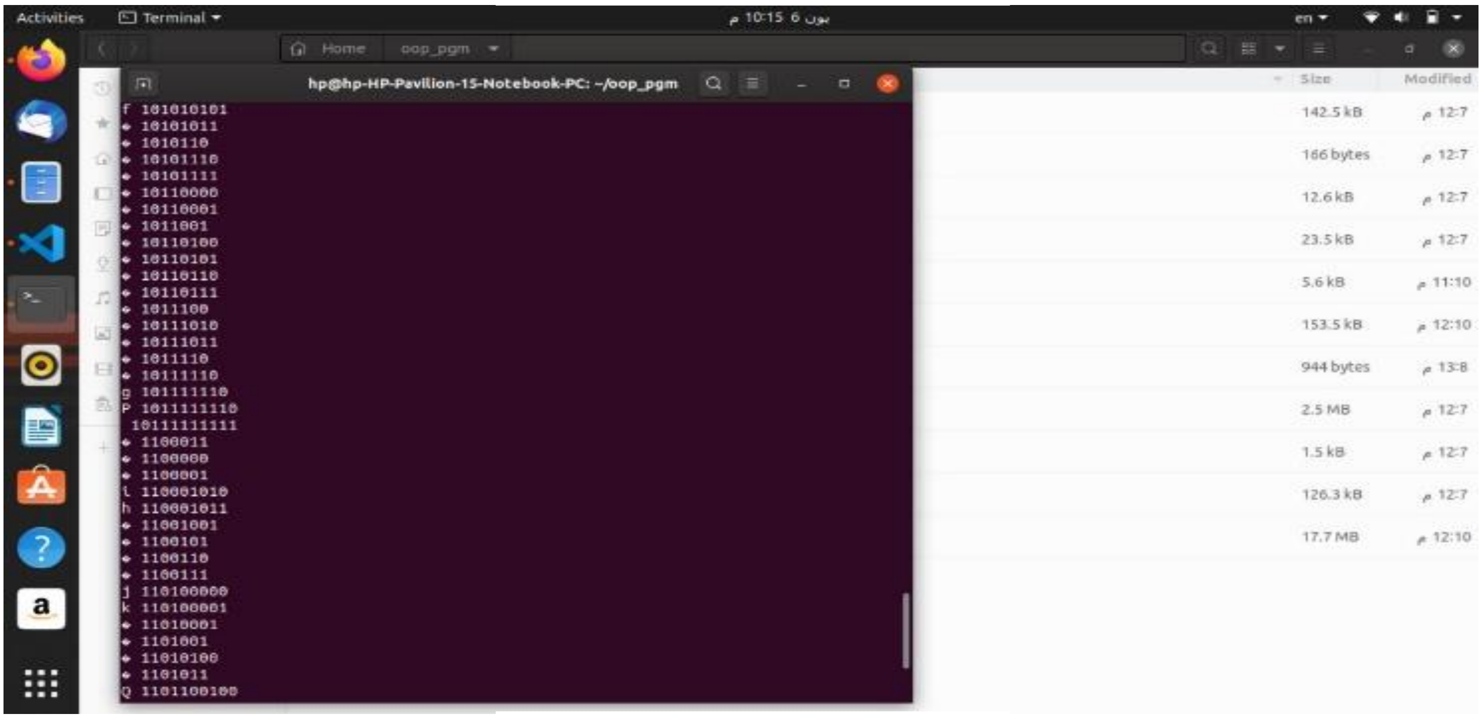
```
P5
1776 1416
255
[illegible]
```

Fig 2: Reading of the image



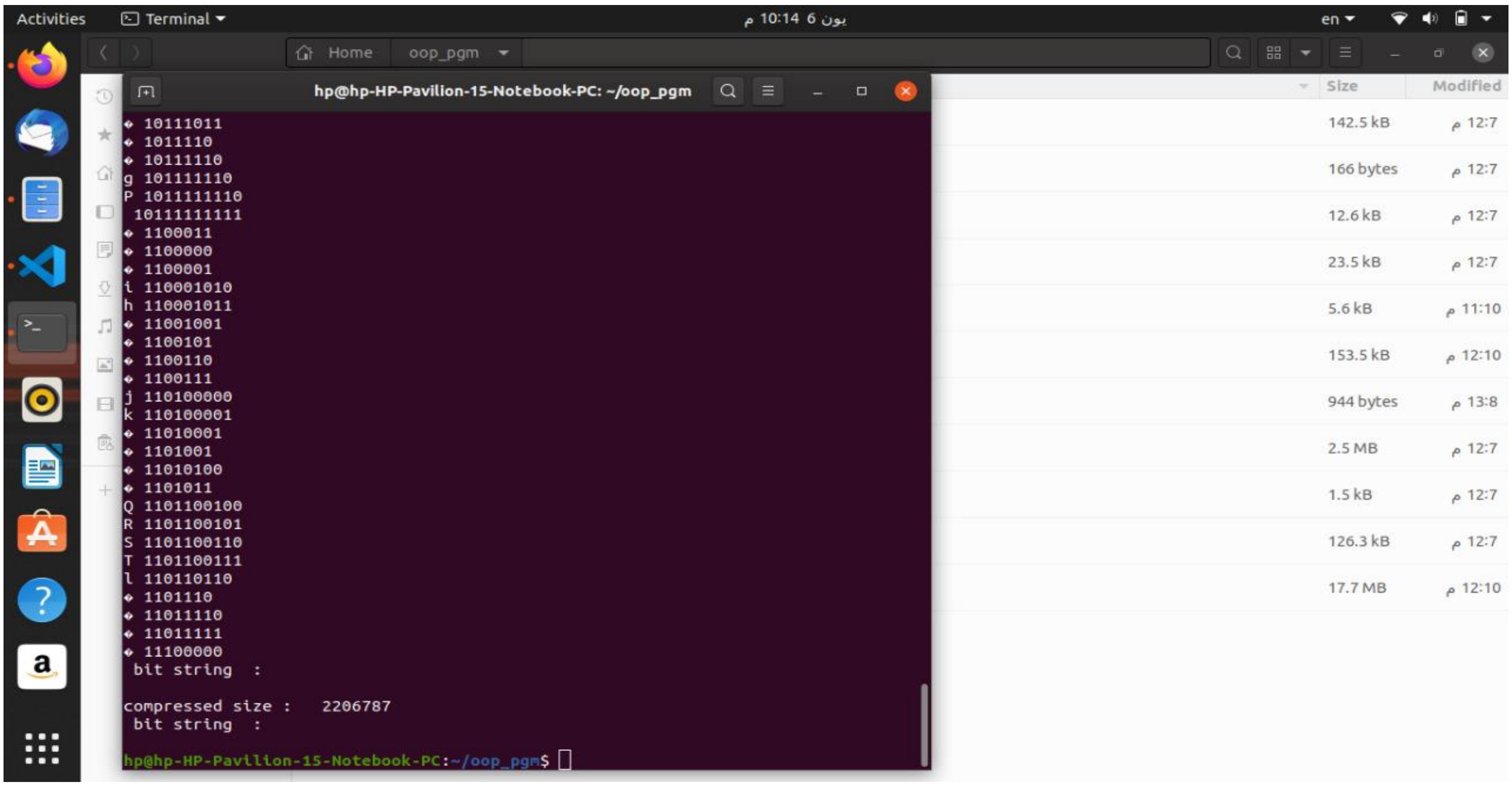
```
hp@hp-HP-Pavilion-15-Notebook-PC:~/oop_pgm$ g++ main.cpp -o img.out
hp@hp-HP-Pavilion-15-Notebook-PC:~/oop_pgm$ ./img.out
opened
num of pixels 2458176
MAXVAL255
all has been read
 Huffman codes are :

N 1111111111
� 111111111011111
� 111111110111101
� 1111111110111001
� 1111111110111000
� 111111111011101
� 111111110111001
� 1111111110110001
� 11111111101100001
� 11111111101100000
� 1111111110101
� 111111111011001
� 111111110110001
� 1111111101100001
� 11111111011000001
� 11111111011000000
� 111111111010
 11111111100
o 111111110
� 11111110
� 11111101
� 11111100
� 1111101
� 1111100
p 111101111
V 1111011101
 1111011100
� 11110110
```

Fig 3 (a): Huffman codes



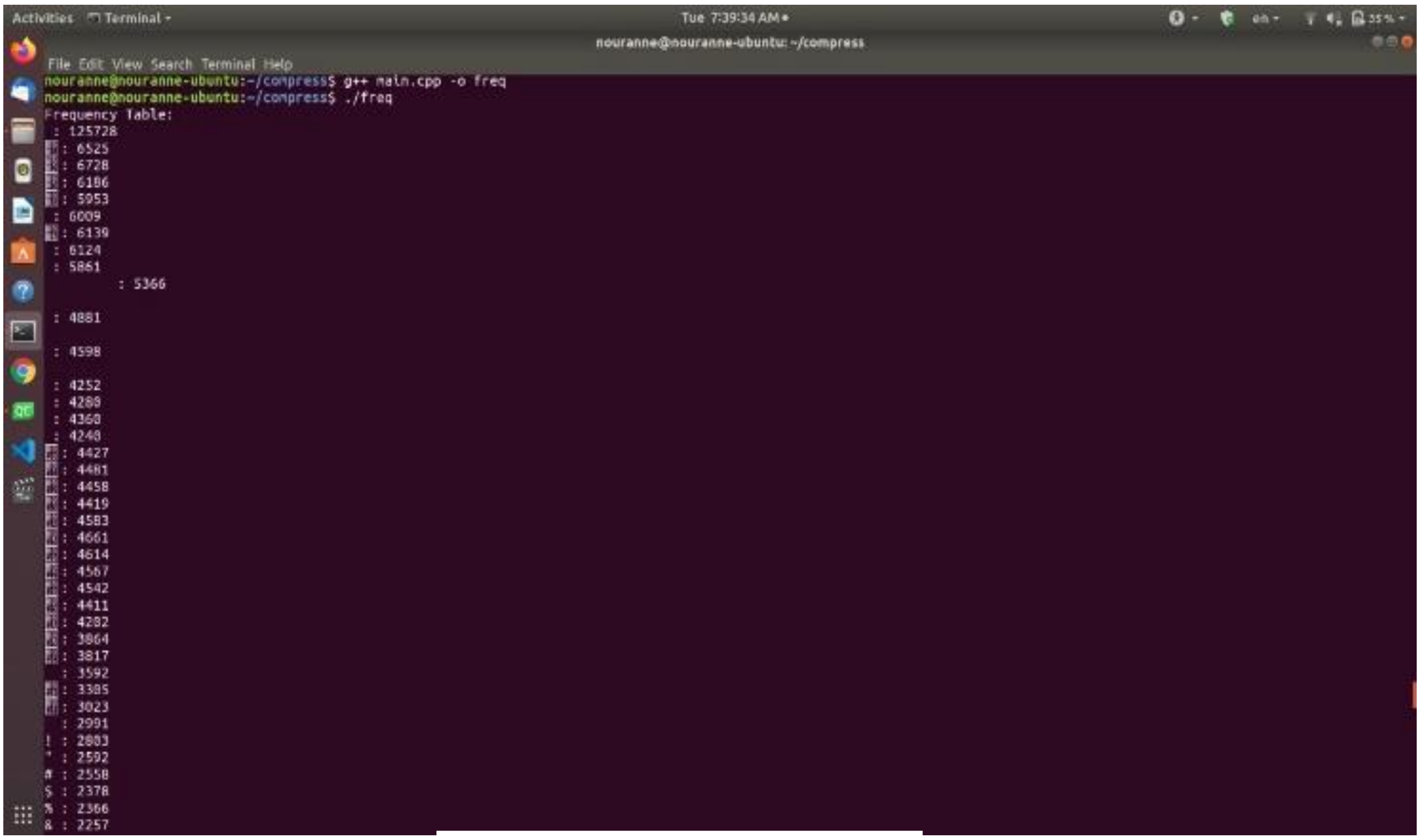
```
f 101010101
� 10101011
� 1010110
� 10101110
� 10101111
� 10110000
� 10110001
� 1011001
� 10110100
� 10110101
� 10110110
� 10110111
� 1011100
� 10111010
� 10111011
� 1011110
� 10111110
g 101111110
P 1011111110
 10111111111
� 1100011
� 1100000
� 1100001
i 110001010
h 110001011
� 11001001
� 1100101
� 1100110
� 1100111
j 110100000
k 110100001
� 11010001
� 1101001
� 11010100
� 1101011
Q 1101100100
```

Fig 3 (b): Huffman codes



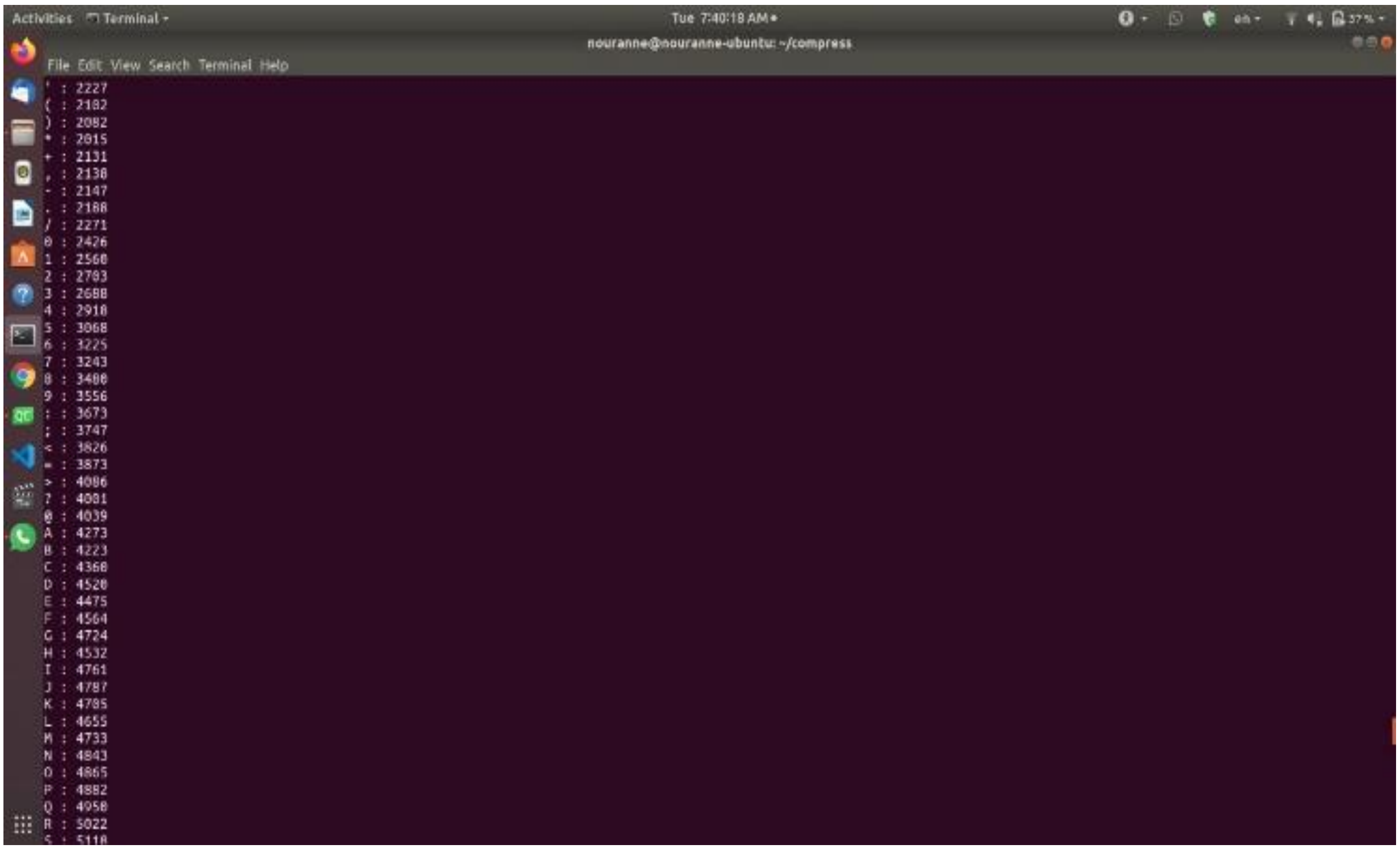
```
hp@hp-HP-Pavilion-15-Notebook-PC: ~/oop_pgm
� 10111011
� 1011110
� 10111110
g 101111110
P 1011111110
 10111111111
� 1100011
� 1100000
� 1100001
i 110001010
h 110001011
� 11001001
� 1100101
� 1100110
� 1100111
j 110100000
k 110100001
� 11010001
� 1101001
� 11010100
� 1101011
Q 1101100100
R 1101100101
S 1101100110
T 1101100111
l 110110110
� 1101110
� 11011110
� 11011111
� 11100000
 bit string  :

compressed size :   2206787
 bit string  :

hp@hp-HP-Pavilion-15-Notebook-PC:~/oop_pgm$ 
```

Fig 3 (c): Huffman codes



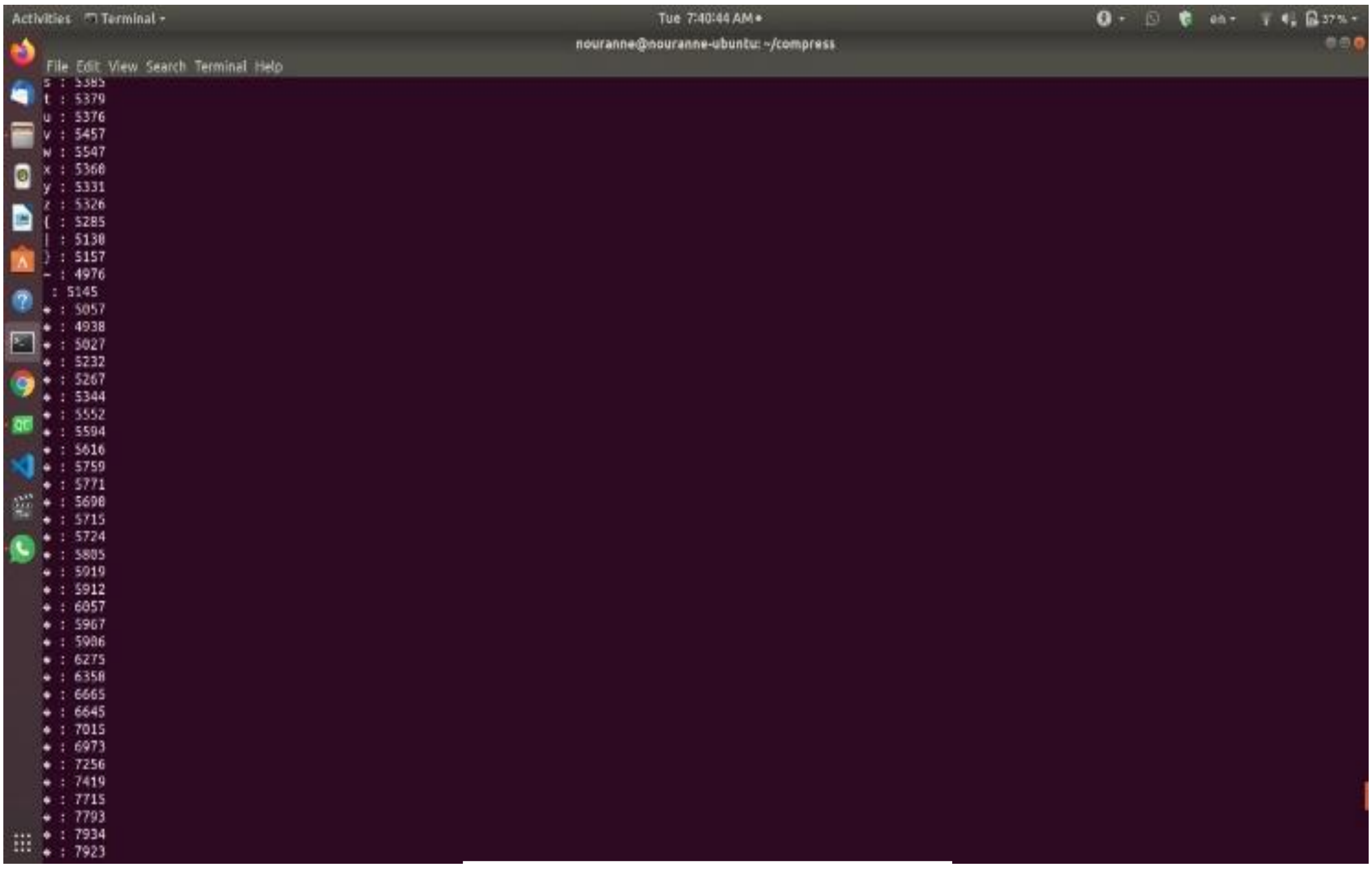
```
nouranne@nouranne-ubuntu: ~/compress
nouranne@nouranne-ubuntu:~/compress$ g++ main.cpp -o freq
nouranne@nouranne-ubuntu:~/compress$ ./freq
Frequency Table:
 : 125728
� : 6525
� : 6728
� : 6186
� : 5953
 : 6009
� : 6139
 : 6124
 : 5861
         : 5366

 : 4881

 : 4598

 : 4252
 : 4289
 : 4360
 : 4248
� : 4427
� : 4481
� : 4458
� : 4419
� : 4583
� : 4661
� : 4614
� : 4567
� : 4542
� : 4411
� : 4282
� : 3864
� : 3817
  : 3592
� : 3385
� : 3023
  : 2991
! : 2803
" : 2592
# : 2558
$ : 2378
% : 2366
& : 2257
```

Fig 4 (a): Huffman frequency table


Activities
Terminal
Tue 7:40:18 AM
nouranne@nouranne-ubuntu: ~/compress
File Edit View Search Terminal Help

```
' : 2227
( : 2182
) : 2082
* : 2015
+ : 2131
, : 2138
- : 2147
. : 2188
/ : 2271
0 : 2426
1 : 2568
2 : 2783
3 : 2688
4 : 2918
5 : 3068
6 : 3225
7 : 3243
8 : 3488
9 : 3556
: : 3673
; : 3747
< : 3826
= : 3873
> : 4086
? : 4081
@ : 4039
A : 4273
B : 4223
C : 4368
D : 4528
E : 4475
F : 4564
G : 4724
H : 4532
I : 4761
J : 4787
K : 4785
L : 4655
M : 4733
N : 4843
O : 4865
P : 4882
Q : 4958
R : 5022
S : 5118
```

Fig 4 (b): Huffman frequency table


Activities
Terminal
Tue 7:40:44 AM
nouranne@nouranne-ubuntu: ~/compress
File Edit View Search Terminal Help

```
s : 5385
t : 5379
u : 5376
v : 5457
w : 5547
x : 5368
y : 5331
z : 5326
{ : 5285
| : 5138
} : 5157
~ : 4976
 : 5145
� : 5057
� : 4938
� : 5027
� : 5232
� : 5267
� : 5344
� : 5552
� : 5594
� : 5616
� : 5759
� : 5771
� : 5698
� : 5715
� : 5724
� : 5805
� : 5919
� : 5912
� : 6057
� : 5967
� : 5986
� : 6275
� : 6358
� : 6665
� : 6645
� : 7015
� : 6973
� : 7256
� : 7419
� : 7715
� : 7793
� : 7934
� : 7923
```

Fig 4 (c): Huffman frequency table

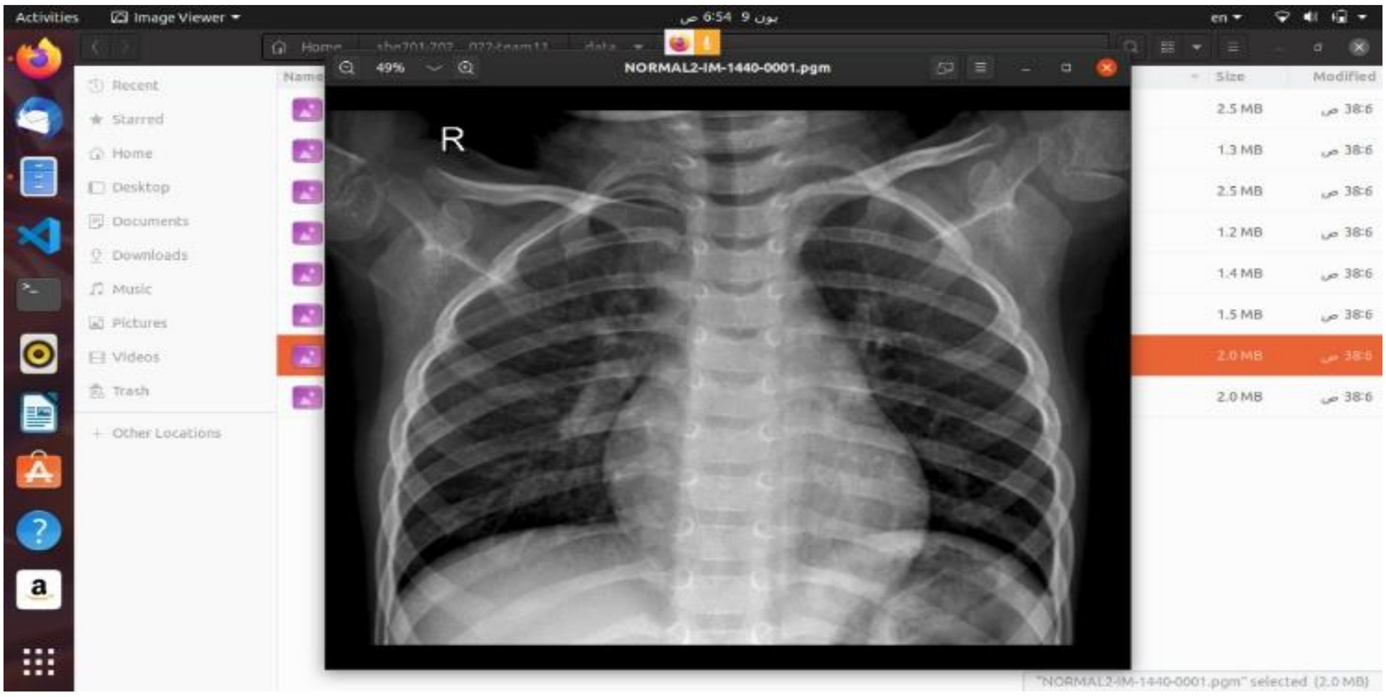


Fig 5: The original image

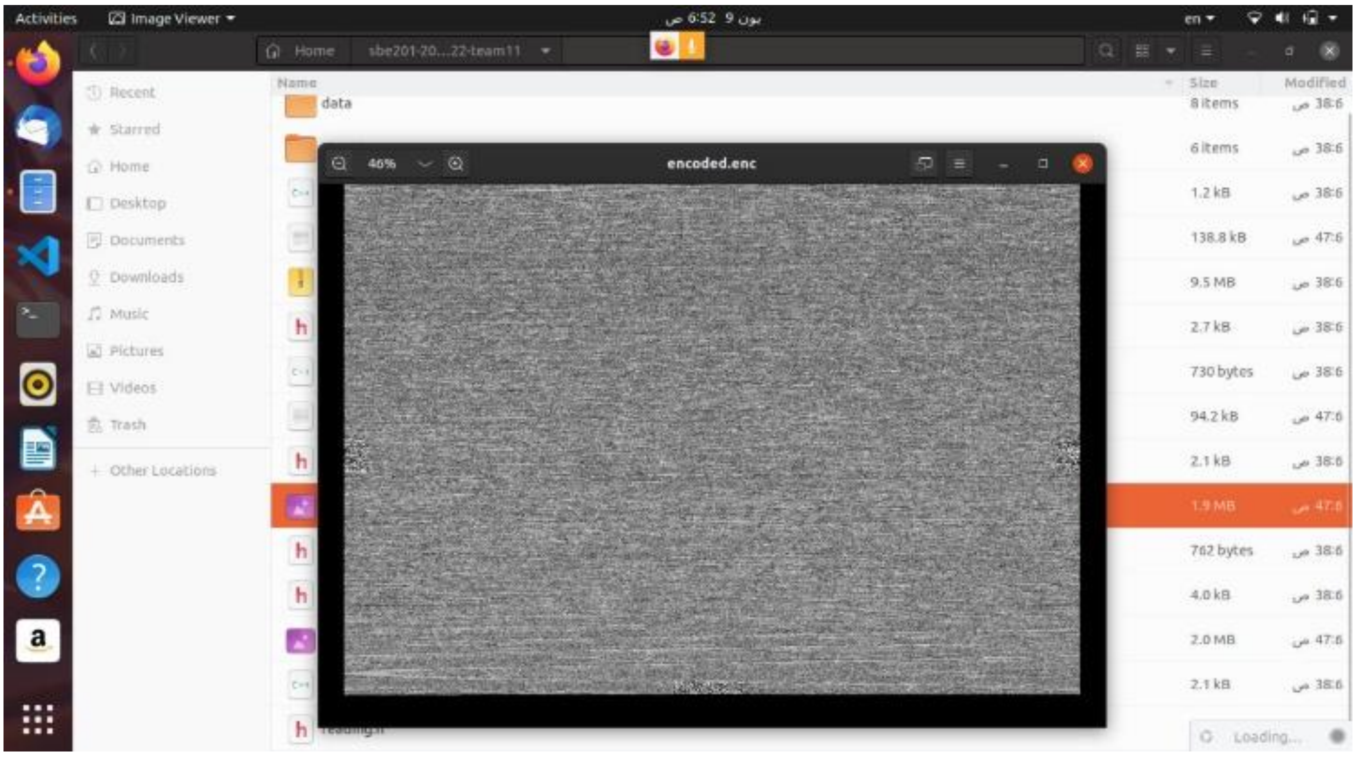
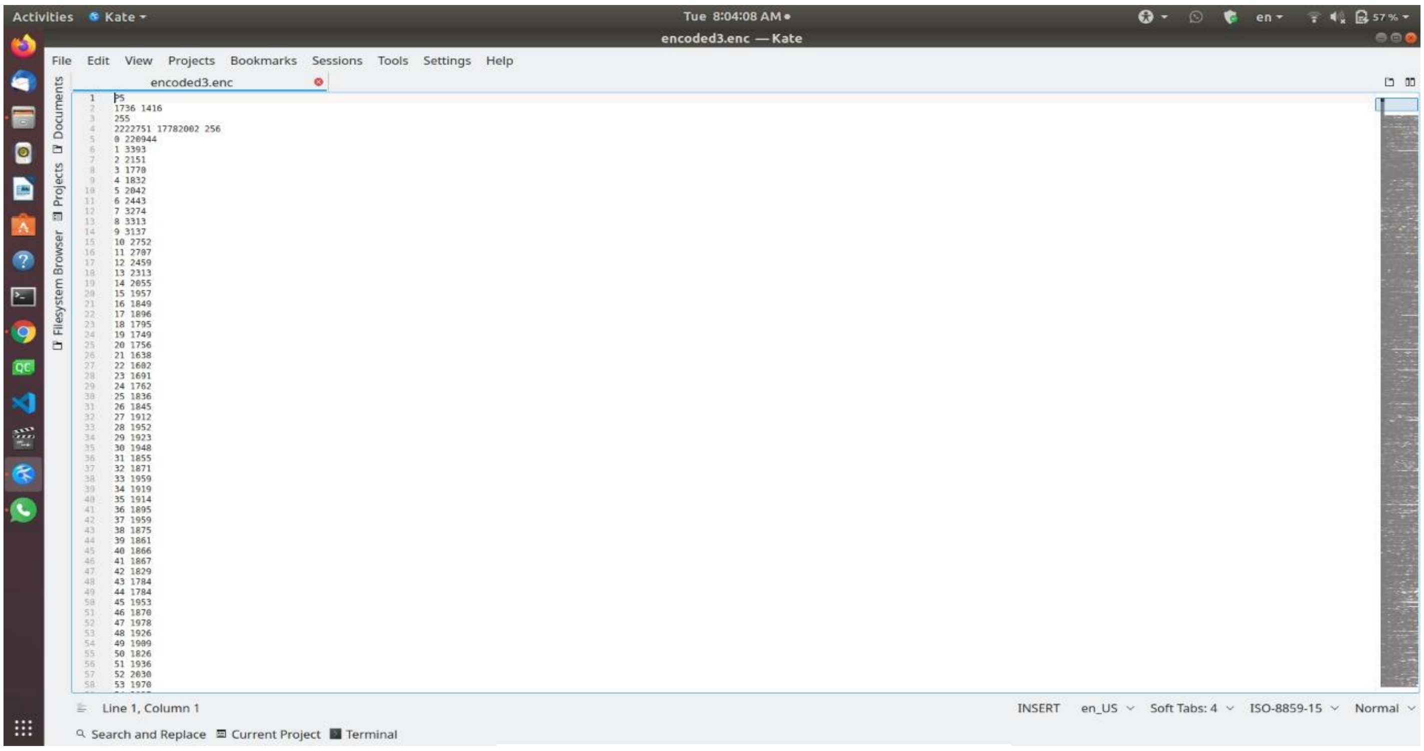


Fig 6: The encoded



```
P5
1736 1416
255
2222751 17782002 256
0 220944
1 3393
2 2151
3 1770
4 1832
5 2042
6 2443
7 3274
8 3313
9 3137
10 2752
11 2787
12 2459
13 2313
14 2055
15 1957
16 1849
17 1896
18 1795
19 1749
20 1756
21 1638
22 1602
23 1691
24 1762
25 1836
26 1845
27 1912
28 1952
29 1923
30 1948
31 1855
32 1871
33 1959
34 1919
35 1914
36 1895
37 1959
38 1875
39 1861
40 1866
41 1867
42 1829
43 1784
44 1784
45 1953
46 1870
47 1978
48 1926
49 1909
50 1826
51 1936
52 2030
53 1970
```

Fig 7: The Frequency map

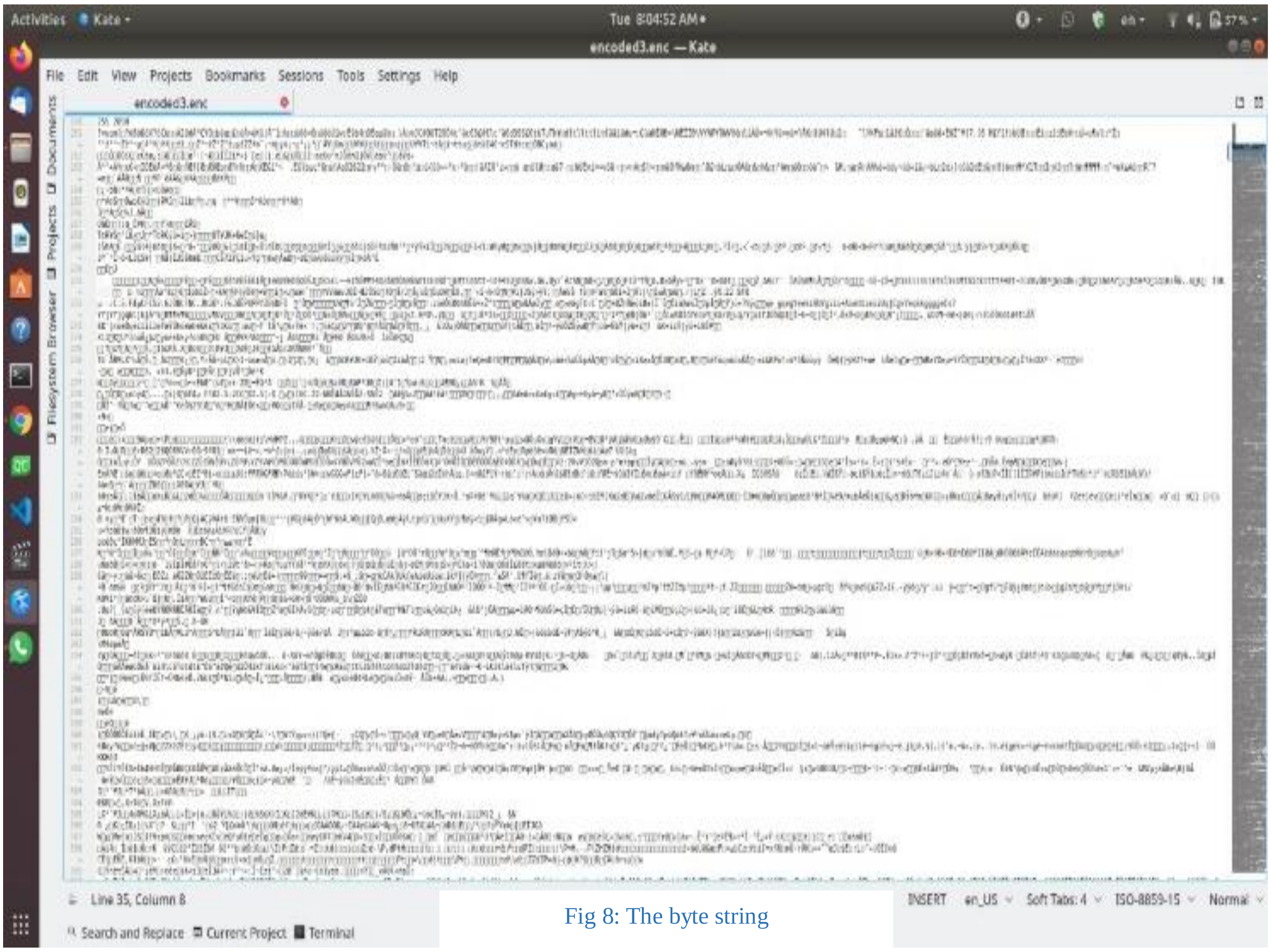


Fig 8: The byte string

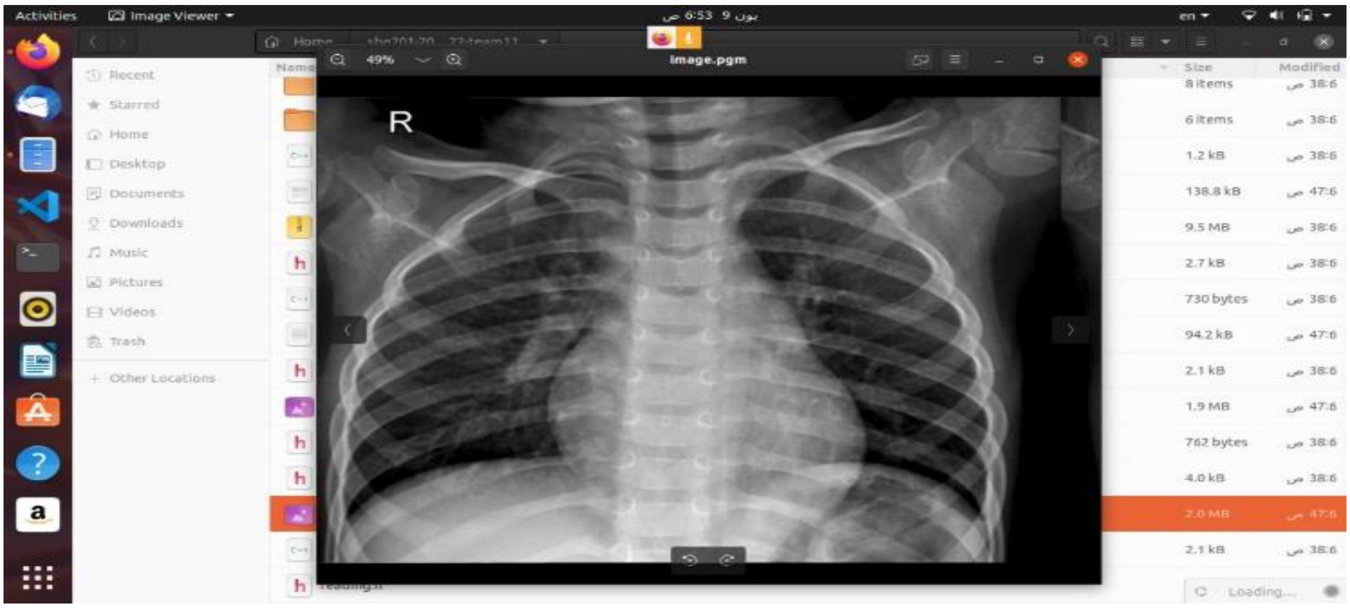
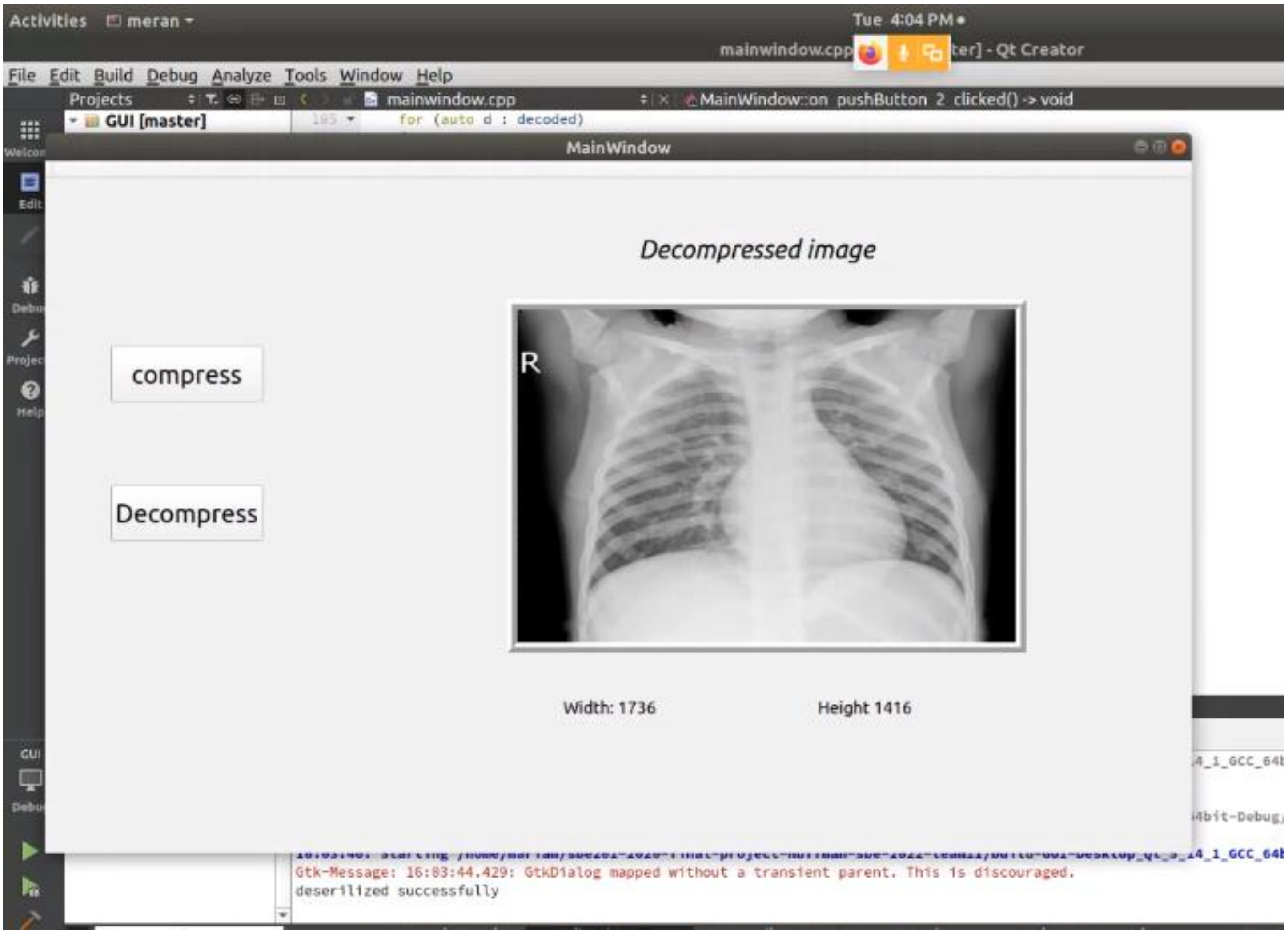


Fig 9: The decoded image



```
Gtk-Message: 16:03:44.429: GtkDialog mapped without a transient parent. This is discouraged.
deserilized successfully
```

Fig 10: GUI

VIII. References

[1] Van Leeuwen, Jan (1976). "On the construction of Huffman trees" (PDF). ICALP: 382–410. Retrieved 2014-02-20.

[2] C. Kuo-Liang, "Efficient Huffman decoding", Information Processing Letters, vol. 61, no. 2, pp. 97-99, 1997. Available: 10.1016/s0020-0190(96)00204-9J.

[3] Y. Lin, S. Huang and C. Yang, "A fast algorithm for Huffman decoding based on a recursion Huffman tree", Journal of Systems and Software, vol. 85, no. 4, pp. 974-980, 2012. Available: 10.1016/j.jss.2011.11.1019

[4] Pi-Chung Wang, Yuan-Rung Yang, Chun-Liang Lee and Hung-Yi Chang, "A Memory-Efficient Huffman Decoding Algorithm", 19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers). Available: 10.1109/aina.2005.33

[5] "vector - C++ Reference", Cplusplus.com, 2020. [Online]. Available: http://www.cplusplus.com/reference/vector/vector/.

[6] S. Singh and V. kumar Gupta, 2016. [Online]. Available: https://www.academia.edu/27803932/JPEG_Image_Compression_and_Decompression_by_Huffman_Coding.

[7] "Reference - C++ Reference", Cplusplus.com . [Online]. Available: http://www.cplusplus.com/reference/ . [Accessed: 06- Jun- 2020].

[8] "vector - C++ Reference", Cplusplus.com, 2020. [Online]. Available: http://www.cplusplus.com/reference/vector/vector/.

[9] "Free Algorithms Book", Books.goalkicker.com. [Online]. Available: https://books.goalkicker.com/AlgorithmsBoo k/.