Project 3 Report

/*Author - Sagar Tewari

UFID - 10488199 */

" I have neither given nor received any unauthorized aid on this assignment ”

Title: Encryption and Data compression of binary instruction set

Approach: The approach deals with 8 bit binary data as the real time systems work on digital data. The data is made secure by using two cryptographic algorithms mentioned as follows -

1. XOR Encryption technique - Taking a 8 bit password from the user and XOR-ing it with the binary data to encrypt it.
2. Caeser cipher algorithm - This algorithm takes a random number input from the user and shifts the data by that amount.

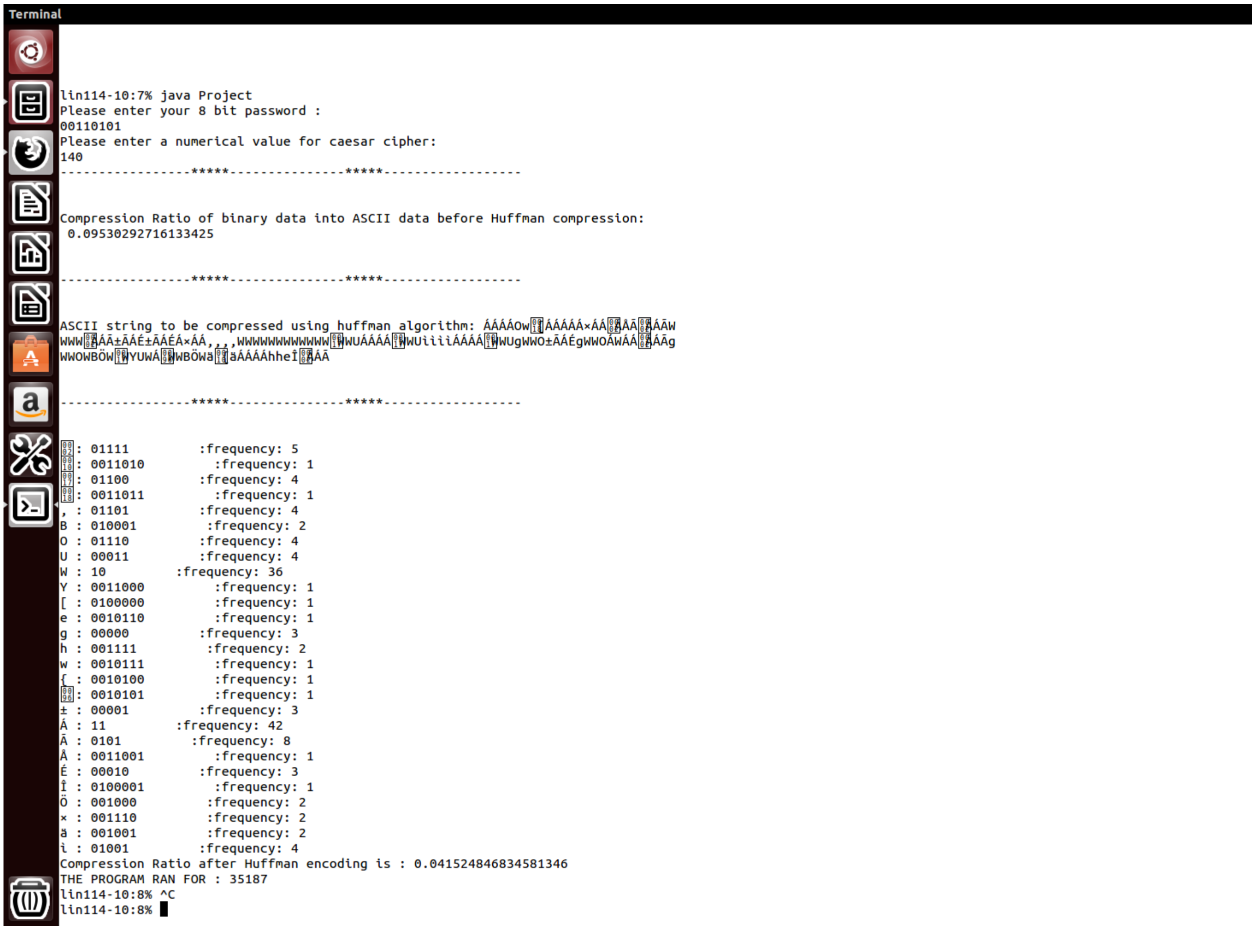
The data now obtained is compressed using the ASCII values corresponding to the binary data we derived and a compression ratio is calculated. As an extension provided to this technique by me, I further compressed the ASCII values using Huffman Encoding technique and computed the compression ratio.

Experimental Setup:

- Software Used - JAVA language
- Metrics measured - Compression Ratio, Runtime
- Input Used - Text file of 8 bit binary values named "inp.txt"
- Compile - javac Project.java
- Run - java Project

Results and Analysis:

Number of binary data	**Input password**	**Shift Value**	**Compression Ratio before Huffman**	**Compression Ratio after Huffman**	**Runtime**
5	11010110	80	0.096153846	0.028846153	15251 ms
5	00110101	140	0.094339622	0.028301886	28721 ms
140	11010110	80	0.097765363	0.042597765	6720 ms
140	00110101	140	0.095302927	0.041524846	35187 ms



```
lin114-10:7% java Project
Please enter your 8 bit password :
00110101
Please enter a numerical value for caesar cipher:
140
-----------------*****---------------*****-----------------


Compression Ratio of binary data into ASCII data before Huffman compression:
 0.09530292716133425


-----------------*****---------------*****-----------------


ASCII string to be compressed using huffman algorithm: ÁÁÁÁOw□ÁÁÁÁÁ×ÁÁ□ÅÁĀ□ÅÁĀW
WWW□ÅÁĀ±ĀÁÉ±ĀÁÉÁ×ÁÁ,,,,WWWWWWWWWWWW□WWUÁÁÁÁ□WWUìììÁÁÁÁ□WWUgWWO±ĀÁÉgWWOÁWÁÁ□ÅÁĀg
WWOWBÖW□WYUWÁ□WWBÖWä□ăÁÁÁÁhheÎ□ÅÁĀ


-----------------*****---------------*****-----------------


□: 01111         :frequency: 5
□: 0011010         :frequency: 1
□: 01100         :frequency: 4
□: 0011011         :frequency: 1
, : 01101         :frequency: 4
B : 010001         :frequency: 2
O : 01110         :frequency: 4
U : 00011         :frequency: 4
W : 10         :frequency: 36
Y : 0011000         :frequency: 1
[ : 0100000         :frequency: 1
e : 0010110         :frequency: 1
g : 00000         :frequency: 3
h : 001111         :frequency: 2
w : 0010111         :frequency: 1
{ : 0010100         :frequency: 1
□: 0010101         :frequency: 1
± : 00001         :frequency: 3
Á : 11         :frequency: 42
Ā : 0101         :frequency: 8
Å : 0011001         :frequency: 1
É : 00010         :frequency: 3
Î : 0100001         :frequency: 1
Ö : 001000         :frequency: 2
× : 001110         :frequency: 2
ä : 001001         :frequency: 2
ì : 01001         :frequency: 4
Compression Ratio after Huffman encoding is : 0.041524846834581346
THE PROGRAM RAN FOR : 35187
lin114-10:8% ^C
lin114-10:8% █
```

Fig 1 : 140 binary inputs

Analysis:

1. The proposed approach was successful in implementing the idea.
2. Huffman encoding successfully exploits the redundancy in the string of ASCII values. It removes replications and each individual character is shown along with its frequency in the table.
3. The binary data is more secure than before.
4. The compression ratio drops significantly from before to after Huffman compression.
5. The runtime for the program just increases fractionally even though the number of inputs were increased tremendously.
6. This project is expected to produce significant results for larger binary inputs i.e. in the order of thousands or lacs as there will be more repetitions of ASCII values in that case.