Algorithm’s final lab

Huffman codes

Ahmed swidan	**5588**
Mostafa talaat	**5401**

Using Huffman code for compression/ decompression of files

Introduction

Huffman code is an algorithm used to decompressed characters by shortening their bits according to their frequency; the higher a character is used, the smaller its code will be. This program allows users to compress text files them decompress them using a Huffman tree containing the character codes.

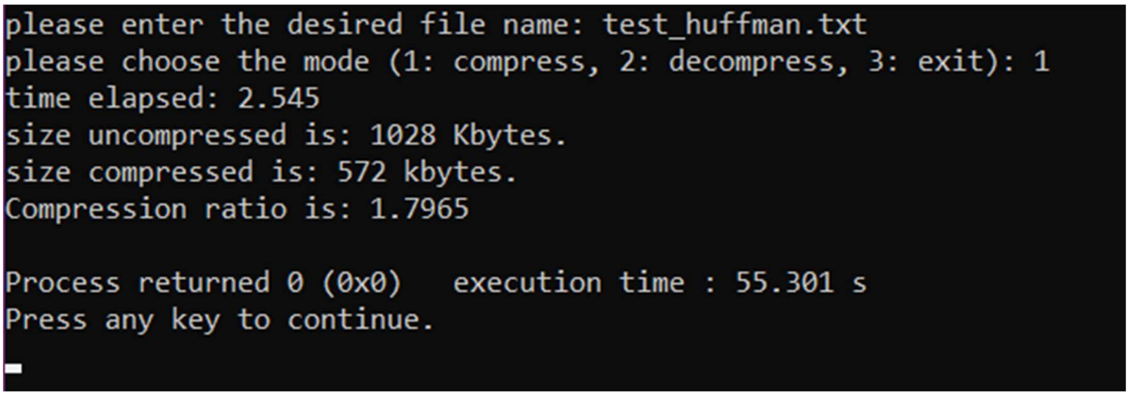
Algorithm

- **Compression**:
  1- Reads the file to count the frequency of every character used.
  2- Forms Huffman tree unique to this file.
  3- Uses the Huffman tree to turn every character to its associated binary code.
  4- Checks if the new string is divisible by 8, if not adds padding (zeros) until it is.
  5- Combines every 8 bits to form one compressed character.
  6- Writes the Huffman tree, padding and compressed text back to the file.
- **Decompression**:
  1- Reads the padding, Huffman tree and the compressed text to the memory.
  2- Changes back every compressed character to 8 bits.
  3- Changers the bits back to characters from the Huffman tree.
  4- Writes the decompressed text back to the file.

Used data structures

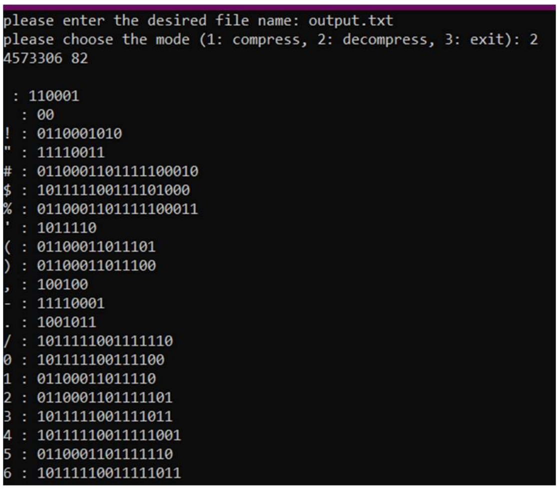
- **Priority queue**: used in the Huffman tree
- **Map**: used to store the frequency of each character
- **Map**: used to store the Huffman code

Test runs



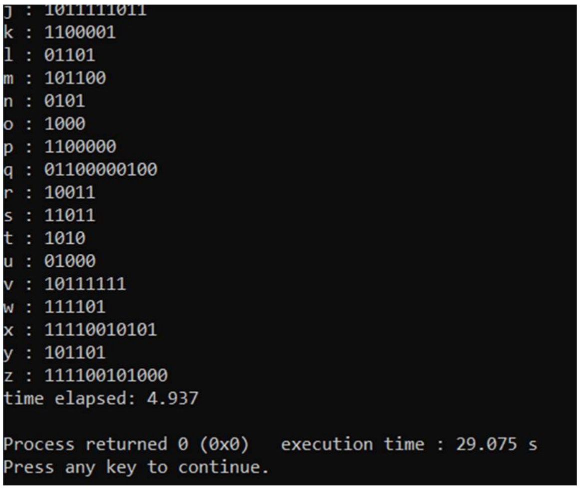
```
please enter the desired file name: test_huffman.txt
please choose the mode (1: compress, 2: decompress, 3: exit): 1
time elapsed: 2.545
size uncompressed is: 1028 Kbytes.
size compressed is: 572 kbytes.
Compression ratio is: 1.7965

Process returned 0 (0x0)   execution time : 55.301 s
Press any key to continue.
```



```
please enter the desired file name: output.txt
please choose the mode (1: compress, 2: decompress, 3: exit): 2
4573306 82

  : 110001
   : 00
! : 0110001010
" : 11110011
# : 0110001101111100010
$ : 101111100111101000
% : 0110001101111100011
' : 1011110
( : 01100011011101
) : 01100011011100
, : 100100
- : 11110001
. : 1001011
/ : 1011111001111110
0 : 101111100111100
1 : 01100011011110
2 : 0110001101111101
3 : 1011111001111011
4 : 10111110011111001
5 : 0110001101111110
6 : 10111110011111011
```



```
j : 1011111011
k : 1100001
l : 01101
m : 101100
n : 0101
o : 1000
p : 1100000
q : 01100000100
r : 10011
s : 11011
t : 1010
u : 01000
v : 10111111
w : 111101
x : 11110010101
y : 101101
z : 111100101000
time elapsed: 4.937

Process returned 0 (0x0)   execution time : 29.075 s
Press any key to continue.
```

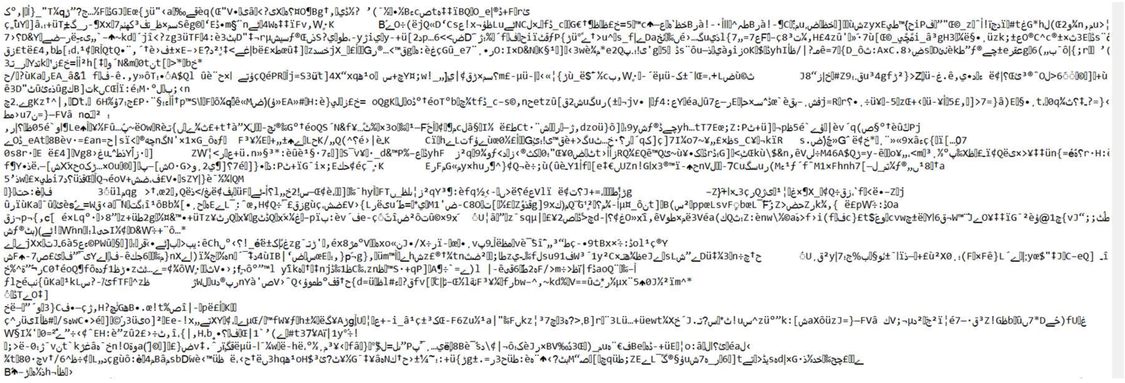


Figure 1 snippet of compressed text