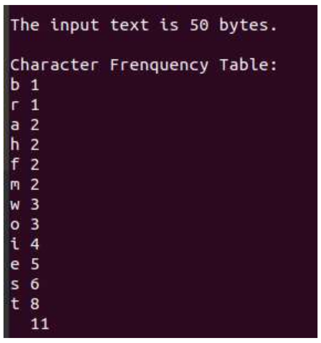
Huffman coding and decoding

We will use the text in source1.txt as an example to show how the Huffman compress works.
The input text is

it was the best of times it was the worst of times

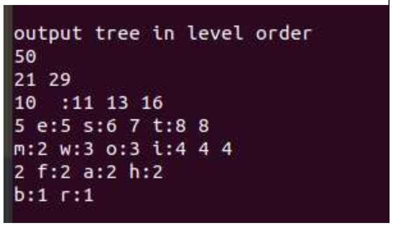
First, run the program with input file and output file.
./huffman source1.txt coded.bin.
The first argument is the input file and the second is the compressed file.
Compress:
Step 1:
Get the frenquencey table:



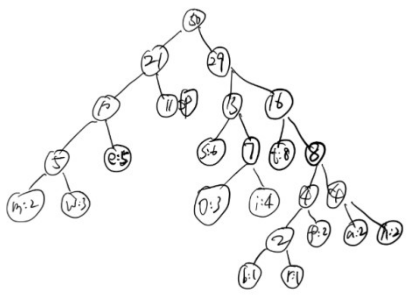
```
The input text is 50 bytes.

Character Frenquency Table:
b 1
r 1
a 2
h 2
f 2
m 2
w 3
o 3
i 4
e 5
s 6
t 8
  11
```

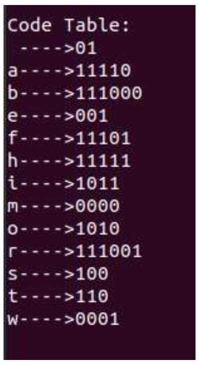
Step 2:
  Push these characters along with their frequency as key to priority_queue.
Step 3:
  Pop two smallest element as two nodes, and construct a parent nodes with these two nodes as childs. New parent node is an internal node, it’s value is the sum of the frequency of its two childs. Then push this new node to priority_queue.
Do the above step again until the queue is empty.
Below is a level order print of the tree,



```
output tree in level order
50
21 29
10  :11 13 16
5 e:5 s:6 7 t:8 8
m:2 w:3 o:3 i:4 4 4
2 f:2 a:2 h:2
b:1 r:1
```



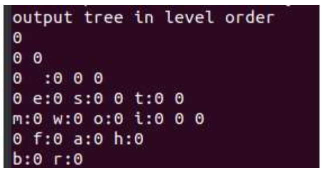
Step 4:
Create a dictionary of codewords: (the first one is space)



```
Code Table:
 ---->01
a---->11110
b---->111000
e---->001
f---->11101
h---->11111
i---->1011
m---->0000
o---->1010
r---->111001
s---->100
t---->110
w---->0001
```

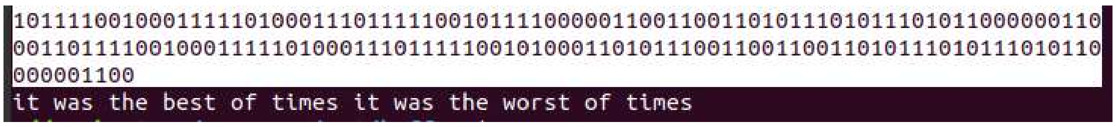
Step 5:
Now we can look up the code table and compress input text as a set of binary bits.Like"1010001".
Step 6:
Store the huffman tree and the coded text as an compressed binary file.
For Huffman tree we don't need the frequency. For internal node, we just write 0x00. For leaf node, we write the character in it. In this way the Huffman tree can be stored as a string. It's a inorder traversal. First check the root, then check root's left tree the go to right tree.

Decompress:
For decompress we need to read the coded file first. Construct Huffman tree from the file.
Note, the tree don't have frequency information.



```
output tree in level order
0
0 0
0  :0 0 0
0 e:0 s:0 0 t:0 0
m:0 w:0 o:0 i:0 0 0
0 f:0 a:0 h:0
b:0 r:0
```

And we can read the coded data from file, too.
To decode, let the coded data be a binary stream and every time we read a bit from it, if it is 1 then we read another bit and go to right tree. If it is 0, we will check next bit in left tree. Once we meet a leaf node, just print the character in it and go back to root. Read one bit each time until there's no more bit in the file.
The coded bit stream of this input is:



```
10111100100011111010001110111110010111100000110011001101011101011101011000000110
00110111100100011111010001110111110010100011010111001100110011010111010111010110
000001100
it was the best of times it was the worst of times
```

And at last we got the original input text.

The sourcefile1 is 50 bytes long and compressed file is 22bytes long.
The sourcefile2 is 504 bytes long and compressed file is 284 bytes long.