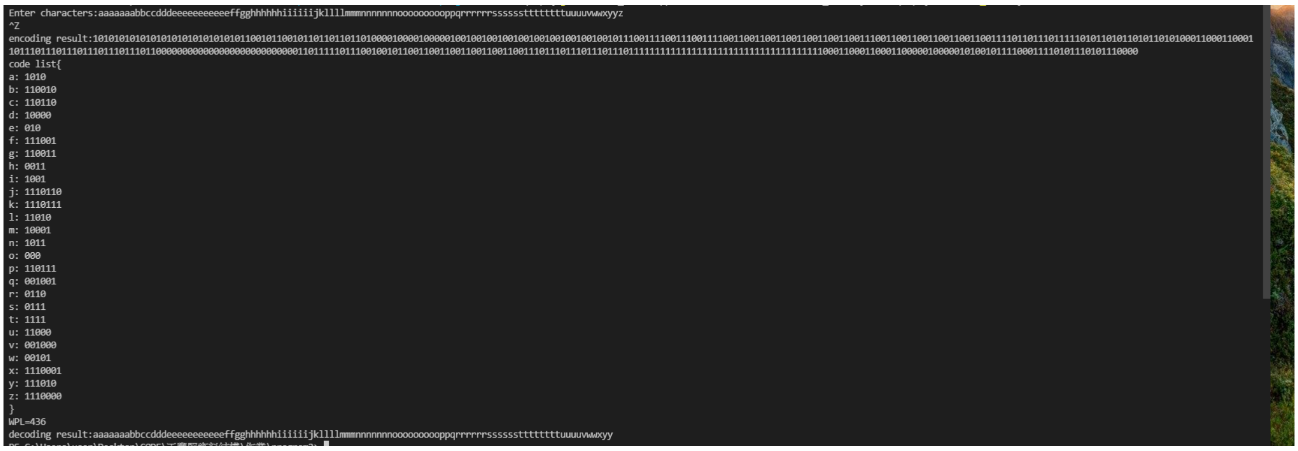
Program assignment 2

109550127 宋哲頤

1.

The idea is to use “variable-length encoding”. We can exploit the fact that some characters occur more frequently than others in a text to design an algorithm that can represent the same piece of text using a lesser number of bits. In variable-length encoding, we assign a variable number of bits to characters depending upon their frequency in the given text. So, some characters might end up taking a single bit, and some might end up taking two bits, some might be encoded using three bits, and so on. The problem with variable-length encoding lies in its decoding.

2.



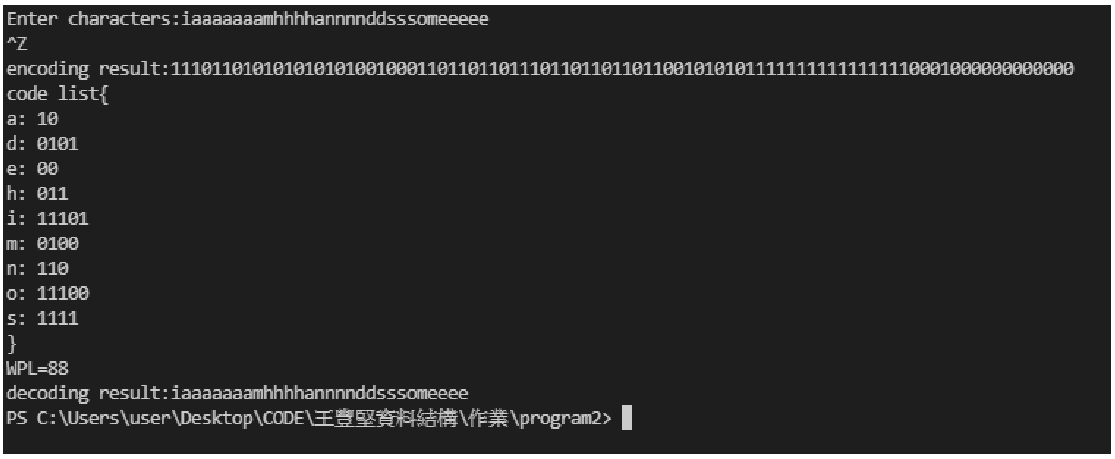
```
Enter characters:aaaaaaabbccdddeeeeeeeeeeeeffgghhhhhhiiiiiijklllllmmmnnnnnnnnoooooooooppqrrrrrrsssssstttttttttuuuuvwwxyyz
^Z
encoding result:1010101010101010101010101010110010110010110110110110100001000010000010010010010010010010010010010111001111001110011110011001100110011001100110011100110011001100110011001111011011101111101011010110101101010001100011000110001
101110111011101110111011000000000000000000000000000110111110111001001011001100110011001100111011101110111011101111111111111111111111111111111111111111100011000110001100000100000101001011111000111101011101011100000
code list{
a: 1010
b: 110010
c: 110110
d: 10000
e: 010
f: 111001
g: 110011
h: 0011
i: 1001
j: 1110110
k: 1110111
l: 11010
m: 10001
n: 1011
o: 000
p: 110111
q: 001001
r: 0110
s: 0111
t: 1111
u: 11000
v: 001000
w: 00101
x: 1110001
y: 111010
z: 1110000
}
WPL=436
decoding result:aaaaaaabbccdddeeeeeeeeeeeeffgghhhhhhiiiiiijklllllmmmnnnnnnnnoooooooooppqrrrrrrsssssstttttttttuuuuvwwxyy
```

characters:aaaaaaabbccdddeeeeeeeeeeeeffgghhhhhhiiiiiijkllllmmmnnnnnnnnoooooo
ooppqrrrrrrsssssstttttttttuuuuvwwxyyz
Encoding
result:10101010101010101010101010101100101100101101101101101000010000 1
00000100100100100100100100100100100101110011110011100111100110011001
10011001100110011100110011001100110011001111011011101111101011010110
1011010100011000110001101110111011101110111011000000000000000000
0000000001101111101110010010110011001100110011001110111011101110
1110111111111111111111111111111111111111111000110001100011000001000001
0100101111100011110101110101110000
code list{
a: 1010 b: 110010 c: 110110 d: 10000 e: 010 f: 111001 g: 110011
h: 0011 i: 1001 j: 1110110 k: 1110111 l: 11010 m: 10001 n: 1011 o: 000

p: 110111   q: 001001   r: 0110   s: 0111   t: 1111   u: 11000   v: 001000
w: 00101   x: 1110001   y: 111010   z: 1110000
}
WPL=436
decoding
result:aaaaaaabbccdddeeeeeeeeeeeffgghhhhhhiiiiiijkllllmmmnnnnnnnnoooooooopp
qrrrrrrsssssstttttttttuuuuvwwxyy

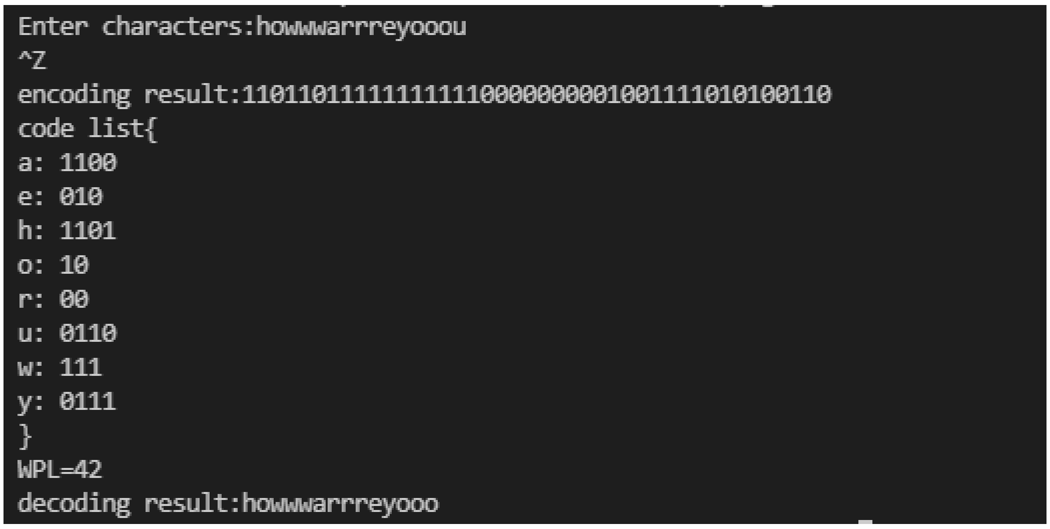
3.

Test1:


```
Enter characters:iaaaaaaamhhhhannnnddsssomeeeee
^Z
encoding result:11101101010101010100100011011011011101101101100101010111111111111111110001000000000000
code list{
a: 10
d: 0101
e: 00
h: 011
i: 11101
m: 0100
n: 110
o: 11100
s: 1111
}
WPL=88
decoding result:iaaaaaaamhhhhannnnddsssomeeeee
PS C:\Users\user\Desktop\CODE\王豊堅資料結構\作業\program2>
```

Enter characters:iaaaaaaamhhhhannnnddsssomeeeee
encoding
result:1110110101010101010010001101101101110110110110010101011111111
1111111110001000000000000
code list{
a: 10 d: 0101 e: 00 h: 011 i: 11101 m: 0100 n: 110 o: 11100 s: 1111
}
WPL=88
decoding result:iaaaaaaamhhhhannnnddsssomeeeee

Test2:



```
Enter characters:howwwarrreyooou
^Z
encoding result:1101101111111111110000000001001111010100110
code list{
a: 1100
e: 010
h: 1101
o: 10
r: 00
u: 0110
w: 111
y: 0111
}
WPL=42
decoding result:howwwarrreyooo
```

Enter characters:howwwarrreyooou
encoding result:1101101111111111110000000001001111010100110
code list{
a: 1100   e: 010   h: 1101   o: 10   r: 00   u: 011   w: 111   y: 0111
}
WPL=42
decoding result:howwwarrreyooo

4.

The technique works by creating a binary tree of nodes. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the character itself, the weight (frequency of appearance) of the character. Internal nodes contain character weight and links to two child nodes. As a common convention, bit 0 represents following the left child, and a bit 1 represents following the right child. A finished tree has n leaf nodes and n-1 internal nodes. It is recommended that Huffman Tree should discard unused characters in the text to produce the most optimal code lengths.

We will use a priority queue for building Huffman Tree, where the node with the lowest frequency has the highest priority. Following are the complete steps:

1. Create a leaf node for each character and add them to the priority queue.
2. While there is more than one node in the queue:

    (1) Remove the two nodes of the highest priority (the lowest frequency) from the queue.

    (2) Create a new internal node with these two nodes as children and a frequency equal to the sum of both nodes' frequencies.

    (3) Add the new node to the priority queue.
3. The remaining node is the root node and the tree is complete.