Name: Karan Kotabagi

University ID: 200217137

Huffman algorithm is used for encoding the character codes in order to compress the text and save the space, in the algorithm which I have implemented in the code, minheap is used i.e. to extract the minimum frequency character and find the two lowest frequency characters and calculate the sum, and again re-insert the node into the minheap. The 0 and 1 are assigned to the left and right edges after the Huffman tree is built and the code is generated for each character by the tree traversal from the root to the corresponding leaf node where the character is stored at the leaf node with the corresponding frequency.

The main important concepts and the functions implemented:

1) There are three functions build_32(PATH), build_64(PATH) and build_128(PATH) which takes the parameters as the path of the file and build the Huffman tree with the specific character considerations.
2) The frequency of the file is calculated by storing the frequency in the freq[256] array as it is declared of the size 256 since there are total 256 ascii codes and the index of the freq[i], I refers to the ascii code that is being stored.
3) Here is the below steps executed in build_32 function:
   - ➔ Calculate the frequency and store the frequency corresponding to the ascii index in the frequency array iff out of 32 characters are encountered in the file, that is declared at the hash map in the calculatefrequency_32 function.

   - ➔ Next the minheap is built using the Heap class that is implemented in the code of max size 256.

   - ➔ The frequencies are assigned to the each node iff the freq[i] is not zero thus to ensure that the freq[i] is not empty since we are storing for 32 characters.

   - ➔ Next the Huffman tree is built based on the Huffman algorithm which takes the heap object as the parameter.

   - ➔ Code is generated and printed using the genCode function and this is recursively implemented explained correctly in the code.

   - ➔ Printotallength function is used to print the bit savings and the other relevant data as shown below in the sample output.

I was able to save accordingly as mentioned below with one of the book Bhagvadgita.

For 32-character comparison, the actual total number of the bits are 645385
Total Bits of all the characters in the Huffman Encoding: 541969
Total Bits Saved: 103416

For 64 bit comparison, the actual total number of the bits are 824418
Total Bits of all the characters in the Huffman Encoding: 624489
Total Bits Saved:
199929

For 128 characters comparison, the actual total number of the bits are 1030414
Total Bits of all the characters in the Huffman Encoding: 699533
Total Bits Saved:
330881

I have implemented the code and have implemented all the three character considerations i.e.. 32, 64 and 128 characters.

I have chosen the book that is attached in the canvas along with the report, and the below is output of the bits that are saved for the each encoding, for the 32 character encoding I have compared it with the 5 bit fixed code length, and for the 64-bit I have compared it with the 6 bit fixed length, and for the 128 characters, I have compared it with the 7 bit fixed code length.

Here is the below output of the program:

Please note that we need to select the appropriate options to see the corresponding output for the 32, 64 or 128 character-code encodings.

Here is the below output:

```
For 32 character encoding enter 1

For 64 character encoding enter 2

For 28 character encoding enter 3

For Exit  0

1

   00 2
h  0100 4
s  0101 4
i  0110 4
n  0111 4
a  1000 4
o  1001 4
d  10100 5
```

```
l  10101 5
t  1011 4
w  110000 6
m  110001 6
c  110010 6
g  110011 6
v  1101000 7
b  1101001 7
f  110101 6
u  110110 6
,  110111 6
e  1110 4
r  11110 5
p  1111100 7
!  11111010 8
q  1111101100 10
z  111110110100 12
?  111110110101 12
x  11111011011 11
'  1111101110 10
j  1111101111 10
.  11111100 8
k  11111101 8
y  1111111 7

For 32-bit comparison, the actual total number of the bits are 645385
Total Bits of all the characters in the Huffman Encoding: 541969
Total Bits Saved: 103416

*********************************** For 64 bit *****************************************

2
   00 2
i  0100 4
n  0101 4
a  0110 4
o  0111 4
y  100000 6
k  1000010 7
v  1000011 7
d  10001 5
w  100100 6
T  1001010 7
j  100101100 9
```

L	100101101	9
D	100101110	9
G	100101111	9
l	10011	5
t	1010	4
m	101100	6
c	101101	6
g	101110	6
S	10111100	8
R	101111010	9
V	1011110110	10
?	10111101110	11
x	10111101111	11
b	1011111	7
f	110000	6
u	110001	6
U	1100100000	10
Y	1100100001	10
F	110010001	9
!	11001001	8
P	110010100	9
W	110010101	9
N	110010110	9
K	1100101110	10
C	1100101111	10
,	110011	6
p	1101000	7
I	11010010	8
A	11010011	8
O	110101000	9
B	110101001	9
.	11010101	8
H	110101100	9
E	110101101	9
q	11010111000	11
Q	1101011100100	13
z	1101011100101	13
J	1101011100110	13
X	1101011100111	13
'	1101011101	10
M	110101111	9
r	11011	5
e	1110	4
h	11110	5

```
s  11111 5
For 64 bit comparison, the actual total number of the bits are 824418
Total Bits of all the characters in the Huffman Encoding: 624489
Total Bits Saved:
199929



**********************************For 128 Character Encoding *************************
3
x  0000000000 10
2  00000000010 11
8  000000000110 12
)  000000000111 12
q  0000000010 10
:  0000000011 10
H  00000001 8
.  0000001 7
E  00000100 8
;  00000101 8
k  0000011 7
y  000010 6
M  00001100 8
'  000011010 9
(  000011011000 12
J  000011011001 12
*  000011011010 12
0  000011011011 12
?  00001101110 11
7  0000110111100 13
9  0000110111101 13
X  000011011111 12
-  0000111 7
s  0001 4
i  0010 4
n  0011 4
a  0100 4

  01010 5
  01011 5
o  0110 4
d  01110 5
v  0111100 7
T  0111101 7
```

w	011111	6
l	10000	5
m	100010	6
j	100011000	9
L	100011001	9
D	100011010	9
G	100011011	9
S	10001110	8
R	100011110	9
V	1000111110	10
6	1000111111000	13
Q	1000111111001	13
5	1000111111010	13
4	1000111111011	13
#	10001111111	11
t	1001	4
c	101000	6
g	101001	6
"	101010000	9
U	1010100010	10
Y	1010100011	10
F	101010010	9
[	10101001100	11
]	10101001101	11
K	1010100111	10
b	1010101	7
f	101011	6
u	101100	6
!	10110100	8
P	101101010	9
W	101101011	9
N	101101100	9
1	10110110100	11
3	101101101010	12
z	1011011010110	13
`	1011011010111000	16
$	1011011010111001	16
%	10110110101110100	17
@	10110110101110101	17
&	1011011010111011	16
/	10110110101111	14
C	1011011011	10
I	10110111	8
,	101110	6

```
p  1011110 7
A  10111110 8
O  101111110 9
B  101111111 9
e  1100 4
r  11010 5
h  11011 5
   111 3
For 128 characters comparison, the actual total number of the bits are 1030414
Total Bits of all the characters in the Huffman Encoding: 699533
Total Bits Saved:
330881
```