```
import heapq
import os

class BinaryTree:
def __init__(self,val,frequency):
self.val=val
self.frequency=frequency
self.right=None
self.left=None

def __lt__(self,temp):
return self.frequency<temp.frequency

def __eq__(self,temp):
return self.frequency==temp.frequency

class HuffmanCode:
def __init__(self,path):
self.path=path# file path for upload and download
self.__arr=[] # container for heap
self.__binary={} # mapping between text and encodings

def __getTextFrequency(self,text):
freqDict={}
for ch in text:
if ch not in freqDict:
freqDict[ch]=0
freqDict[ch]+=1
return freqDict

def __buildHeap(self,freq):
for key in freq:
node=BinaryTree(key,freq[key])
heapq.heappush(self.__arr,node)

def __buildBinaryTree(self):
```

```
while len(self.__arr)>1:
node1=heapq.heappop(self.__arr)
node2=heapq.heappop(self.__arr)
supernode=BinaryTree(None,node1.frequency+node2.frequency)
supernode.right=node2
supernode.left=node1
heapq.heappush(self.__arr,supernode)
return

def __getBinHelper(self,root,bits):
#base case
if root==None:
return
if root.value is not None:
# leaf node is reached
self.__binary[root.value]=bits
return
#recursive case
#moving left
self.__getBinHelper(self,root.left,bits+'0')
#moving right
self.__getBinHelper(self,root.right,bits+'1')

def __getBinaryCodeFromTree(self):
root=heapq.heappop(self.__arr)
self.__getBinHelper(self,root,'')

def __encode(self,text):
temp=''
for ch in text:
temp+=self.__binary[ch]
return temp

# now as the data will be stored in bits of 8 so we need to add some padding/bits of zeros towards end of t
def __getPaddedCode(self,encodedText):
padding=8-len(encodedText)%8
```

```
for i in range(padding):
encodedText+='0'
paddingInfo="{0:08b}".format()
finalCode=paddingInfo+encodedText
return finalCode

def __convertToBytes(self,paddedText):
temp=[]
for i in range(0,len(paddedText),8):
byteArr=paddedText[i:i+8]
temp.append(byteArr)
return temp

def fileCompress(self):
fileName,fileExtension=os.path.splitext(self.path)
outputPath=fileName+'.bin'
with open(self.path,'r+') as file,open(outputPath,'wb') as output:
text=file.read()
text=text.rstrip()
freq=self.__getTextFrequency(text)
build_heap=self.__buildHeap(freq)
self.__buildBinaryTree()
self.__getBinaryCodeFromTree()
encoded_text=self.__encode(text)
padded_text=self.__getPaddedCode(encoded_text)
byte_arr=self.__convertToBytes(padded_text)
#padding the encoded text
byteData=bytes(byte_arr)
output.write(byteData)
print("compressed Successfuly")

path=input('Enter the path')
h=HuffmanCode(path)
h.fileCompress()
```

# To access the file and extract text out of the file
# Create frequency of each text and store it in dictionary
# Use min heap to get the top two elements with minimum frequency
# Construct the binary tree using from headp
# Contruct code from binary tree and store it in dictionary
# Construct the encoded text
# Return the binary file as an output

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dol

Section 1.10.32 of "de Finibus Bonorum et Malorum", written by Cicero in 45 BC
"Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam

1914 translation by H. Rackham
"But I must explain to you how all this mistaken idea of denouncing pleasure and praising pain was born ar

Section 1.10.33 of "de Finibus Bonorum et Malorum", written by Cicero in 45 BC
"At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum delenit

1914 translation by H. Rackham
"On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and dem