```python
# based on lengths of assigned codes based on frequencies
# variable length codes: prefix codes
import os
import time
import math
import random
import sys
from btree import *


class Huffman:
    def __init__(self):
        print('Name of input file in relative directory (input.txt)?: ',
              end='',
              flush=True)
        # f_name = str(input())
        f_name = 'input.txt'
        print()
        # access input file
        self.dir_path = os.path.dirname(__file__)
        input_file_path = os.path.join(self.dir_path, f_name)
        self.f_table = []
        self.chars = []
        self.char_count = 0
        self.char_map = {}
        with open(input_file_path) as f:
            c = f.read(1)
            while c:
                self.chars.append(c)
                self.char_count += 1
                index = self.find_existing_char(c)
                if index == -1:  # char does not exist in frequency table
                    self.f_table.append(Node(c, 1))
                    self.char_map[c] = self.f_table[-1]
                else:  # char already exists in frequency table
                    self.f_table[index].freq += 1
                c = f.read(1)
        self.f_table_len = len(self.f_table)
        self.visited = [0 for i in range(len(self.f_table))]
        print('No steps or animations (0),\nAnimate (1),\nJust steps (2)?: ',
              end='',
              flush=True)
        self.animate = int(input())
        self.trees = []

        while not self.everything_visited():
            self.insert_lowest()
        self.print_f_table()
        self.decode(self.encode())

        os.system('perl -ne \'print pack("B2", $_)\' < encoded_message.txt >
encoded_message.bin')

    def find_existing_char(self, c):
        if len(self.f_table) == 0:
            return -1
        for index, char in enumerate(self.f_table):
            if c == char.char:
                return index
        return -1
```

```
60
61      def insert_lowest(self):
62          self.print_not_visited()
63          if self.animate == 1:
64              input()
65
66          l = None
67          l_index = -1
68          for i, v in enumerate(self.visited):
69              if v == 0:
70                  if(l is None or
71                     l is not None and
72                     self.f_table[i].freq < l.freq):
73                      l = self.f_table[i]
74                      l_index = i
75          self.visited[l_index] = 1
76
77          h = None
78          h_index = -1
79          for i, v in enumerate(self.visited):
80              if v == 0:
81                  if(h is None or
82                     h is not None and
83                     self.f_table[i].freq < h.freq and
84                     self.f_table[i].freq >= l.freq):
85                      h = self.f_table[i]
86                      h_index = i
87          self.visited[h_index] = 1
88
89          self.current_tree = HuffmanBinaryTree([], self.animate)
90          root = Node(l.freq + h.freq, l.freq + h.freq)
91          self.f_table.append(root)
92          self.visited.append(0)
93          self.current_tree.insert(root, l, h)
94          self.current_tree.print_tree()
95
96      def encode(self):
97          encoded_message = []
98          for char in self.chars:
99              print(char, end='', flush=True)
100             encoded_message.append(self.char_map[char].code)
101         print('\n' + str(encoded_message))
102         output_file_path = os.path.join(self.dir_path, 'encoded_message.txt')
103         with open(output_file_path, 'w') as f:
104             for code in encoded_message:
105                 #f.write(str(code) + '\n')
106                 f.write(str(code))
107         return encoded_message
108
109     def decode(self, encoded_message):
110         print('Decoding encoded message using tree ... ')
111         decoded_message = ''
112         for code in encoded_message:
113             decoded_message += str(
114                 self.current_tree.find_char(
115                     self.current_tree.get_root(),
116                     code))
117         print(decoded_message)
118         output_file_path = os.path.join(self.dir_path, 'decoded_message.txt')
119         with open(output_file_path, 'w') as f:
```

```
120             for char in decoded_message:
121                 f.write(char)
122 
123     def everything_visited(self):
124         count = 0
125         for v in self.visited:
126             if v == 0:
127                 count += 1
128         if count == 1:
129             return True
130         else:
131             return False
132 
133     def print_not_visited(self):
134         print('chars/roots left: ', end='', flush=True)
135         for i, v in enumerate(self.visited):
136             if v == 0:
137                 if self.f_table[i].get_char() == '\n':
138                     print('\\n', end='', flush=True)
139                 elif self.f_table[i].get_char() == '\t':
140                     print('\\t', end='', flush=True)
141                 else:
142                     print(self.f_table[i].get_char(), end='', flush=True)
143                 print(':' +
144                       str(self.f_table[i].freq) +
145                       ' | ',
146                       end='',
147                       flush=True)
148         print()
149 
150     def print_f_table(self):
151         for i, node in enumerate(self.f_table):
152             if i < self.f_table_len:
153                 print(node.char_info())
154 
155 Huffman()
156 
```