```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-

import heapq
import os
import sys
from collections import defaultdict

class HeapNode:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None


    def __lt__(self, other):
        return self.freq < other.freq


class HuffmanCoding:
    def __init__(self, path):
        self.path = path
        self.heap = []
        self.codes = {}
        self.reverse_mapping = {}


    # functions for compression:

    def make_frequency_dict(self, text):
        freq_dict = defaultdict(int)
        for char in text:
            freq_dict[char] += 1
        return freq_dict

    def make_heap(self, freq_dict):
        for key in freq_dict:
            node = HeapNode(key, freq_dict[key])
            heapq.heappush(self.heap, node)

    def merge_nodes(self):
        while len(self.heap) > 1:
            left = heapq.heappop(self.heap)
            right = heapq.heappop(self.heap)

            parent = HeapNode(None, left.freq + right.freq)
```

```
        parent.left = left
        parent.right = right
        heapq.heappush(self.heap, parent)

    def make_codes_helper(self, root, current_code):
        if root is None:
            return

        if root.char is not None:
            self.codes[root.char] = current_code
            self.reverse_mapping[current_code] = root.char
            return

        self.make_codes_helper(root.left, current_code + '0')
        self.make_codes_helper(root.right, current_code + '1')

    def make_codes(self):
        root = heapq.heappop(self.heap)
        current_code = ''
        self.make_codes_helper(root, current_code)

    def get_encoded_text(self, text):
        encoded_text = ''
        for char in text:
            encoded_text += self.codes[char]
        return encoded_text

    def pad_encoded_text(self, encoded_text):
        extra_padding = 8 - len(encoded_text) % 8
        for i in range(extra_padding):
            encoded_text += '0'

        padded_info = '{0:08b}'.format(extra_padding)
        padded_encoded_text = padded_info + encoded_text
        return padded_encoded_text

    def get_bytearray(self, padded_encoded_text):
        if len(padded_encoded_text) % 8 != 0:
            print('Encoded text not padded properly')
            exit(0)

        b = bytearray()
        for i in range(0, len(padded_encoded_text), 8):
            byte = padded_encoded_text[i: i+8]
            b.append(int(byte, base=2))
        return b
```

```
def compress(self):
    filename, file_ext = os.path.splitext(self.path)
    output_path = filename + '.huf'

    with open(self.path, 'r+') as file, \
        open(output_path, 'wb') as output:

        text = file.read()

        freq_dict = self.make_frequency_dict(text)
        self.make_heap(freq_dict)
        self.merge_nodes()
        self.make_codes()

        encoded_text = self.get_encoded_text(text)
        padded_encoded_text = self.pad_encoded_text(encoded_text)

        b = self.get_bytearray(padded_encoded_text)
        output.write(bytes(b))

    print('Compressed')
    return output_path



""" functions for decompdression: """

def remove_padding(self, padded_encoded_text):
    padded_info = padded_encoded_text[:8]
    extra_padding = int(padded_info, base=2)

    padded_encoded_text = padded_encoded_text[8:]
    encoded_text = padded_encoded_text[: -extra_padding]

    return encoded_text

def decode_text(self, encoded_text):
    current_code = ''
    decoded_text = ''

    for bit in encoded_text:
        current_code += bit
        if current_code in self.reverse_mapping:
            char = self.reverse_mapping[current_code]
            decoded_text += char
            current_code = ''
```

```
        return decoded_text


    def decompress(self, input_path):
        filename, file_ext = os.path.splitext(self.path)
        output_path = filename + '_decompressed' + '.txt'


        with open(input_path, 'rb') as file, \
                open(output_path, 'w') as output:
            bit_str = ''

            byte = file.read(1)
            while byte != b'':
                byte = ord(byte)
                bits = bin(byte)[2:].rjust(8, '0')
                bit_str += bits
                byte = file.read(1)

            encoded_text = self.remove_padding(bit_str)
            decompressed_text = self.decode_text(encoded_text)

            output.write(decompressed_text)

        print('Decompressed')
        return output_path


FILE_PATH = sys.argv[1]

h = HuffmanCoding(FILE_PATH)

compressed_path = h.compress()
print('compressed_path: ', compressed_path)
decompressed_path = h.decompress(compressed_path)
print('decompressed_path: ', decompressed_path)
```

压缩结果比较：

103K Nov 23 18:42 Aesop_Fables.huf
186K Nov 23 17:39 Aesop_Fables.txt

909K Nov 23 18:44 graph.huf
2.0M Nov 23 18:43 graph.txt