```
# -*- coding: windows-1251 -*-
import string
import math
import heapq
from collections import Counter, namedtuple

f = open(r"res/1.txt", "r", encoding="windows-1251")
text = f.read()
text = text.lower()


def remove_chars_from_text(txt, chars):
    return ''.join([ch for ch in txt if ch not in chars])


spec_chars = string.punctuation + '«»–…1234567890—
qwertyuiopasdfghjklzxcvbnm\n\t\xa0©'
text = remove_chars_from_text(text, spec_chars)
text = text.replace('ё', 'е')
text = text.replace('ъ', 'ь')

text_tokens = dict()


def shannon(txt, n):
    for d in range(1, n):
        for i in range(len(txt) - d + 1):
            if text_tokens.get(txt[i:i + d]) is None:
                text_tokens[txt[i:i + d]] = 1
            else:
                text_tokens[txt[i:i + d]] = text_tokens.get(txt[i:i + d]) + 1
        if d == 1:
            max_h = math.log(len(text_tokens), 2)
            print("max_h = ", max_h)
        h = 0
        summa = sum(text_tokens.values())
        for value in text_tokens.values():
            h += (-1) * (value / summa) * math.log(value / summa, 2)
        h /= d
        print(f"h{d} = {h}")
        text_tokens.clear()


class Node(namedtuple("Node", ["first", "second", "third"])):
    def walk(self, code, acc):
        self.first.walk(code, acc + "0")
        self.second.walk(code, acc + "1")
        self.third.walk(code, acc + "2")
```

```
class Leaf(namedtuple("Leaf", ["char"])):
    def walk(self, code, acc):
        code[self.char] = acc or "0"


def huffman_encode(s):
    h = []
    for ch, freq in Counter(s).items():
        h.append((freq, len(h), Leaf(ch)))
    heapq.heapify(h)
    count = len(h)
    if count % 2 == 0:
        h.append((0, 0, Leaf("\n")))
    while len(h) > 1:
        freq1, _count1, first = heapq.heappop(h)
        freq2, _count2, second = heapq.heappop(h)
        freq3, _count3, third = heapq.heappop(h)
        heapq.heappush(h, (freq1 + freq2 + freq3, count, Node(first, second,
third)))
        count += 1
    code = {}
    if h:
        [(_freq, _count, root)] = h
        root.walk(code, "")
    return code


def shannon1(txt):
    for i in range(len(txt) - 1):
        if text_tokens.get(txt[i:i + 1]) is None:
            text_tokens[txt[i:i + 1]] = 1
        else:
            text_tokens[txt[i:i + 1]] = text_tokens.get(txt[i:i + 1]) + 1
    h = 0
    summa = sum(text_tokens.values())
    for value in text_tokens.values():
        h += (-1) * (value / summa) * math.log(value / summa, 2)
    return h / 1


def main():
    code = huffman_encode(text)
    # for ch in sorted(code.items(), key=lambda item: len(item[1])):
    #     if ch[0] == '\n':
    #         continue
    #     print("{}: {}".format(ch[0], ch[1]))
    encode_text = "".join(code[ch] for ch in text)
    tokens = Counter(text)
    summa = sum(tokens.values())
    sr_dl = 0
    for ch in sorted(code):
        sr_dl += tokens[ch] / summa * len(code[ch])
    print("Средня длина кодового слова: {}".format(sr_dl))
    shannon(encode_text, 4)


main()
```