Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)
Институт информатики и вычислительной техники
Кафедра прикладной математики и кибернетики

Практическая работа №2
по дисциплине «Теория информации»
на тему «Побуквенное кодирование текстов»

Выполнили:
студенты гр.ИП-014
Обухов А.И.

Проверила:
Старший преподаватель каф. ПМиК
Дементьева Кристина Игоревна

Новосибирск 2024 г.

Цель работы: Экспериментальное изучение избыточности сжатия текстового файла.

Язык программирования: C, C++, C#, Python

Результат: программа, тестовые примеры, отчет.

Задание:

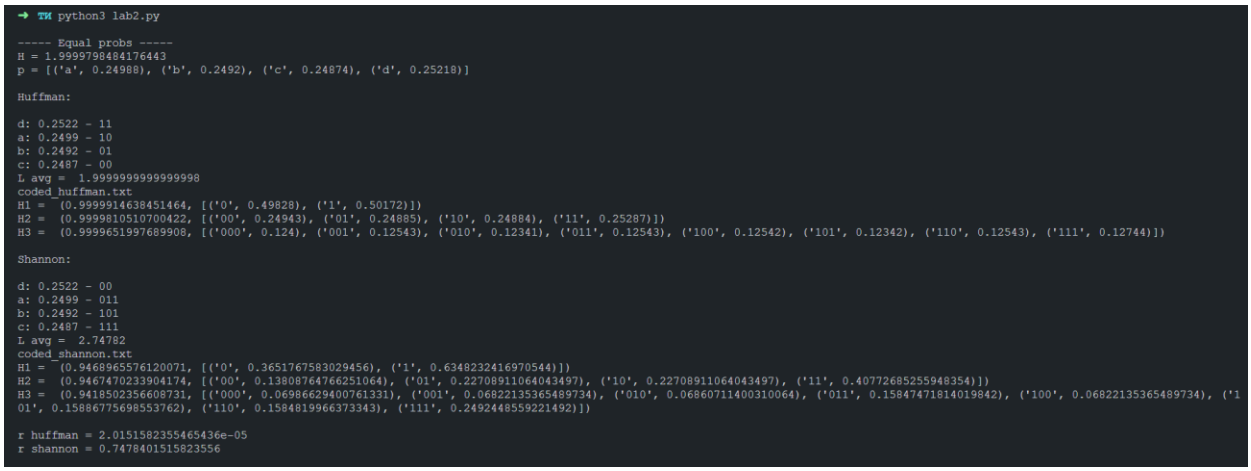
1. Запрограммировать процедуру двоичного кодирования текстового файла побуквенным кодом. В качестве методов сжатия использовать метод Хаффмана и метод Шеннона (или метод Фано). Текстовые файлы использовать те же, что и в практической работе 1.

2. Вычислить среднюю длину кодовых слов и оценить избыточность кодирования для каждого построенного побуквенного кода.

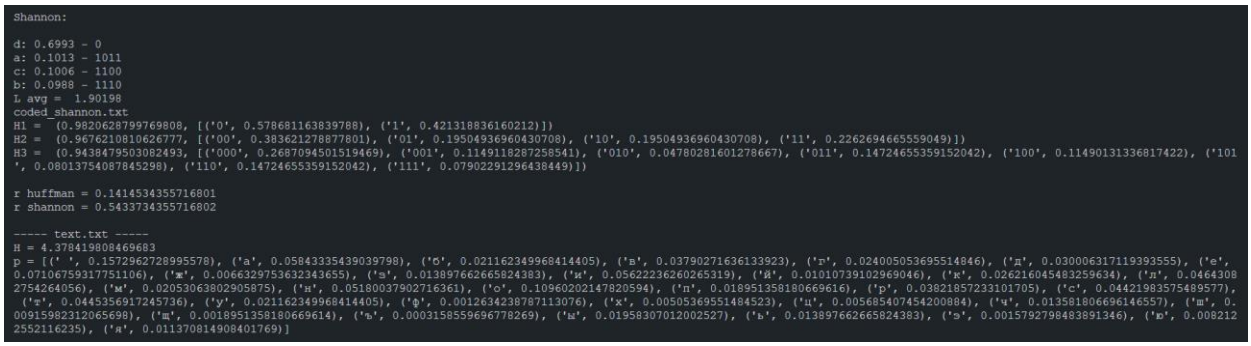
3. После кодирования текстового файла вычислить оценки энтропии файла с закодированным текстом H_1, H_2, H_3 (после кодирования последовательность содержит 0 и 1) и заполнить таблицу.

Метод кодирования	Название текста	Оценка избыточности кодирования	H_1	H_2	H_3
Метод Хаффмана	text.txt	0.02595	0.99669	0.99669	0.99664
Метод Шеннона	text.txt	0.4421	0.99001	0.98942	0.98796

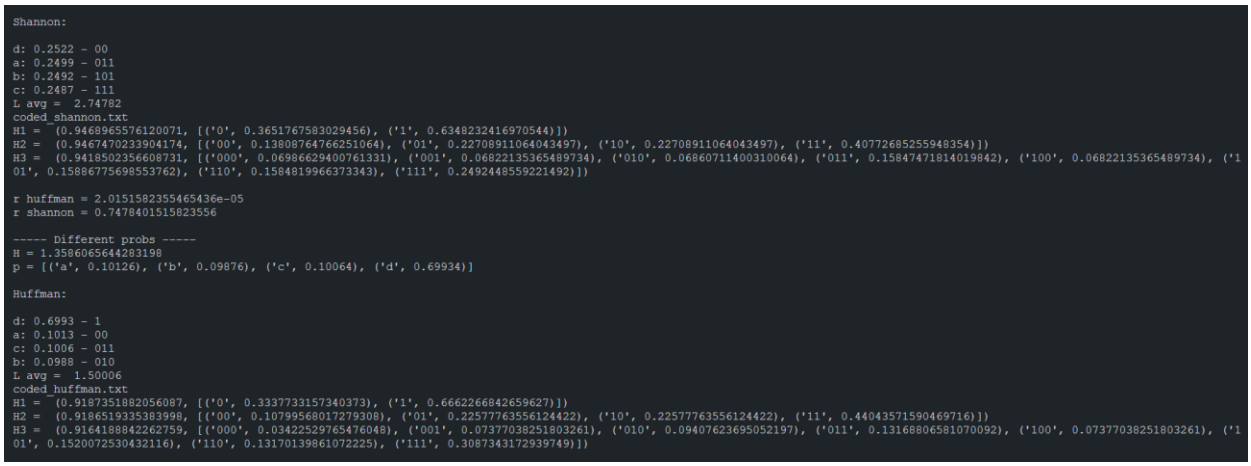
Скриншоты работы программы:



```
➜ ти python3 lab2.py

----- Equal probs -----
H = 1.9999798484176443
p = [('a', 0.24988), ('b', 0.2492), ('c', 0.24874), ('d', 0.25218)]

Huffman:

d: 0.2522 - 11
a: 0.2499 - 10
b: 0.2492 - 01
c: 0.2487 - 00
L avg =  1.9999999999999998
coded_huffman.txt
H1 =  (0.9999914638451464, [('0', 0.49828), ('1', 0.50172)])
H2 =  (0.9999810510700422, [('00', 0.24943), ('01', 0.24885), ('10', 0.24884), ('11', 0.25287)])
H3 =  (0.9999651997689908, [('000', 0.124), ('001', 0.12543), ('010', 0.12341), ('011', 0.12543), ('100', 0.12542), ('101', 0.12342), ('110', 0.12543), ('111', 0.12744)])

Shannon:

d: 0.2522 - 00
a: 0.2499 - 011
b: 0.2492 - 101
c: 0.2487 - 111
L avg =  2.74782
coded_shannon.txt
H1 =  (0.9468965576120071, [('0', 0.3651767583029456), ('1', 0.6348232416970544)])
H2 =  (0.9467470233904174, [('00', 0.1380876476625106 4), ('01', 0.22708911064043497), ('10', 0.22708911064043497), ('11', 0.40772685255948354)])
H3 =  (0.9418502356608731, [('000', 0.06986629400761331), ('001', 0.06822135365489734), ('010', 0.06860711400310064), ('011', 0.15847471814019842), ('100', 0.0682213536548973 4), ('101', 0.15886775698553762), ('110', 0.158481996637334 3), ('111', 0.24924485592214 92)])

r huffman = 2.0151582355465436e-05
r shannon = 0.7478401515823556
```



```
Shannon:

d: 0.6993 - 0
a: 0.1013 - 1011
c: 0.1006 - 1100
b: 0.0988 - 1110
L avg =  1.90198
coded_shannon.txt
H1 =  (0.9820628799769808, [('0', 0.578681163839788), ('1', 0.421318836160212)])
H2 =  (0.9676210810626777, [('00', 0.383621278877801), ('01', 0.19504936960430708), ('10', 0.19504936960430708), ('11', 0.2262694665559049)])
H3 =  (0.9438479503082493, [('000', 0.2687094501519469), ('001', 0.1149118287258541), ('010', 0.04780281601278667), ('011', 0.14724655359152042), ('100', 0.11490131336817422), ('101', 0.08013754087845298), ('110', 0.14724655359152042), ('111', 0.07902291296438449)])

r huffman = 0.1414534355716801
r shannon = 0.5433734355716802

----- text.txt -----
H = 4.378419808469683
p = [(' ', 0.1572962728995578), ('а', 0.05843335439039798), ('б', 0.021162349968414405), ('в', 0.03790271636133923), ('г', 0.024005053695514846), ('д', 0.030006317119393555), ('е', 0.07106759317751106), ('ж', 0.006632975363234365 5), ('з', 0.013897662665824383), ('и', 0.056222362602653 19), ('й', 0.01010739102969046), ('к', 0.026216045483259634), ('л', 0.046430827542640 56), ('м', 0.020530638029058 75), ('н', 0.051800379027163 61), ('о', 0.109602021478205 94), ('п', 0.018951358180669616), ('р', 0.038218572331017 05), ('с', 0.044219835754895 77), ('т', 0.044535691724573 6), ('у', 0.021162349968414405), ('ф', 0.001263423878711 3076), ('х', 0.005053695514845 23), ('ц', 0.005685407454200884), ('ч', 0.013581806696146557), ('ш', 0.009159823120656 98), ('щ', 0.001895135818066961 4), ('ъ', 0.000315855969778269), ('ы', 0.019583070120025 27), ('ь', 0.013897662665824383), ('э', 0.001579279848389 1346), ('ю', 0.008212255211623 5), ('я', 0.011370814908401769)]
```



```
Shannon:

d: 0.2522 - 00
a: 0.2499 - 011
b: 0.2492 - 101
c: 0.2487 - 111
L avg =  2.74782
coded_shannon.txt
H1 =  (0.9468965576120071, [('0', 0.3651767583029456), ('1', 0.6348232416970544)])
H2 =  (0.9467470233904174, [('00', 0.1380876476625106 4), ('01', 0.22708911064043497), ('10', 0.22708911064043497), ('11', 0.4077268525594835 4)])
H3 =  (0.9418502356608731, [('000', 0.06986629400761331), ('001', 0.06822135365489734), ('010', 0.06860711400310064), ('011', 0.15847471814019842), ('100', 0.0682213536548973 4), ('101', 0.15886775698553762), ('110', 0.158481996637334 3), ('111', 0.24924485592214 92)])

r huffman = 2.0151582355465436e-05
r shannon = 0.7478401515823556

----- Different probs -----
H = 1.3586065644283198
p = [('a', 0.10126), ('b', 0.09876), ('c', 0.10064), ('d', 0.69934)]

Huffman:

d: 0.6993 - 1
a: 0.1013 - 00
c: 0.1006 - 011
b: 0.0988 - 010
L avg =  1.50006
coded_huffman.txt
H1 =  (0.9187351882056087, [('0', 0.3337733157340373), ('1', 0.6662266842659627)])
H2 =  (0.9186519335383998, [('00', 0.1079956801727930 8), ('01', 0.2257776355612442 2), ('10', 0.2257776355612442 2), ('11', 0.4404357159046971 6)])
H3 =  (0.9164188842262759, [('000', 0.0342252976547604 8), ('001', 0.0737703825180326 1), ('010', 0.0940762369505219 7), ('011', 0.13168806581070092), ('100', 0.0737703825180326 1), ('101', 0.1520072530432116), ('110', 0.1317013986107222 5), ('111', 0.3087343172939749)])
```

```
Huffman:

 : 0.1573 - 110
о: 0.1096 - 010
е: 0.0711 - 1010
а: 0.0584 - 1000
и: 0.0562 - 0111
н: 0.0518 - 0011
л: 0.0464 - 0001
т: 0.0445 - 0000
с: 0.0442 - 11111
р: 0.0382 - 10111
в: 0.0379 - 10110
д: 0.0300 - 10010
к: 0.0262 - 01100
г: 0.0240 - 00101
б: 0.0212 - 111101
у: 0.0212 - 111100
м: 0.0205 - 111011
ы: 0.0196 - 111001
п: 0.0190 - 111000
ь: 0.0139 - 100110
э: 0.0139 - 011011
ч: 0.0136 - 011010
я: 0.0114 - 001000
й: 0.0101 - 1110101
ш: 0.0092 - 1001111
ю: 0.0082 - 1001110
ж: 0.0066 - 0010011
ц: 0.0057 - 0010010
х: 0.0051 - 11101000
щ: 0.0019 - 111010010
э: 0.0016 - 1110100110
ф: 0.0013 - 11101001111
ъ: 0.0003 - 11101001110
L avg =  4.41156328490207
coded_huffman.txt
H1 =  (0.9954309563793968, [('0', 0.46022767953032145), ('1', 0.5397723204696785)])
H2 =  (0.9952875554448601, [('00', 0.20720269205985536), ('01', 0.252953390133887), ('10', 0.2530249874704661), ('11', 0.28674733299921246)])
H3 =  (0.9948471666469795, [('000', 0.09952029784492017), ('001', 0.10768239421493521), ('010', 0.11706164530679458), ('011', 0.13589174482709243), ('100', 0.10768239421493521), ('1
01', 0.14527099591895182), ('110', 0.13589174482709243), ('111', 0.15085558817212)])
```

```
Shannon:

 : 0.1573 - 000
о: 0.1096 - 0010
е: 0.0711 - 0110
а: 0.0584 - 01010
и: 0.0562 - 01100
н: 0.0518 - 01110
л: 0.0464 - 11111
т: 0.0445 - 10001
с: 0.0442 - 10011
р: 0.0382 - 10100
в: 0.0379 - 10101
д: 0.0300 - 101101
к: 0.0262 - 101111
г: 0.0240 - 111110
б: 0.0212 - 110010
у: 0.0212 - 110110
м: 0.0205 - 110101
ы: 0.0196 - 110110
п: 0.0190 - 111011
ь: 0.0139 - 1110010
э: 0.0139 - 1110100
ч: 0.0136 - 1110110
я: 0.0114 - 1111001
й: 0.0101 - 1111001
ш: 0.0092 - 1111010
ю: 0.0082 - 1111100
ж: 0.0066 - 11111010
ц: 0.0057 - 11111011
х: 0.0051 - 11111101
щ: 0.0019 - 1111111010
э: 0.0016 - 1111111100
ф: 0.0013 - 1111111110
ъ: 0.0003 - 11111111110
L avg =  4.924826279216676
coded_shannon.txt
H1 =  (0.9981847492941696, [('0', 0.47492303745510517), ('1', 0.5250769625448948)])
H2 =  (0.9967385633498957, [('00', 0.20985120574653668), ('01', 0.2650076962544895), ('10', 0.2650718317085685), ('11', 0.2600051308363263)])
```

Анализ результатов работы программы

В ходе выполнения практической работы были проведены эксперименты по вычислению оценок энтропии Шеннона текстов с использованием различных методов побуквенного кодирования. Для этого были использованы три текстовых файла с разными свойствами, включая последовательность символов с равными вероятностями, последовательность символов с различными вероятностями и художественный текст на русском языке.

Были реализованы три метода оценки энтропии Шеннона:

1. Первый метод (H1) оценивает энтропию по отдельным символам файла.
2. Второй метод (H2) оценивает энтропию по парам символов.
3. Третий метод (H3) оценивает энтропию по тройкам символов.

Для каждого текстового файла были вычислены указанные оценки энтропии для каждого метода. Результаты показали, что значения оценок энтропии различаются в зависимости от метода и свойств текста. В частности, метод Хаффмана обеспечивает более оптимальное сжатие по сравнению с методом Шеннона для данного текста "text.txt".

Также были рассмотрены значения избыточности кодирования для обоих методов. Полученные результаты позволяют сделать вывод о более эффективном использовании битовых ресурсов при кодировании с использованием метода Хаффмана.

Таким образом, выполнение практической работы позволило изучить основные свойства энтропии Шеннона и сравнить различные методы ее оценки. Полученные результаты могут быть полезны для анализа

эффективности и качества методов побуквенного кодирования при работе с текстовыми данными.

Листинг программы

```
import heapq
from math import ceil
from lab1 import *

class Node:
    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

def huffman_encode(line: str) -> float:
    print("\nHuffman:\n")

    def build_huffman_tree(text):
        char_freq = Counter(text)
        heap = [Node(char, freq) for char, freq in char_freq.items()]
        heapq.heapify(heap)

        while len(heap) > 1:
            left = heapq.heappop(heap)
            right = heapq.heappop(heap)
            merged = Node(None, left.freq + right.freq)
            merged.left = left
            merged.right = right
            heapq.heappush(heap, merged)

        return heap[0]

    def build_huffman_codes(node, prefix="", codes={}):
        if node:
            if node.char is not None:
                codes[node.char] = prefix
            build_huffman_codes(node.left, prefix + "0", codes)
            build_huffman_codes(node.right, prefix + "1", codes)

    split_line = list(line[i: i + 1] for i in range(len(line)))
    probabilities = {k: v / len(split_line) for k, v in
Counter(split_line).items()}
    probabilities = dict(sorted(probabilities.items(), key=lambda item: item[1],
reverse=True))

    root = build_huffman_tree(line)
    codes = {}
    build_huffman_codes(root, "", codes)
```

```
    for i in probabilities.keys():
        print(f"{i}: {probabilities[i]:.4f} - {codes[i]}")
    l_average = sum(probabilities[i] * len(codes[i]) for i in
probabilities.keys())
    print("L avg = ", l_average)

    print("coded_huffman.txt")
    with open("./output/coded_huffman.txt", "w") as f:
        for i in line:
            f.write(codes[i])

    with open("./output/coded_huffman.txt", "r") as f:
        text = f.readline()
    for i in range(1, 4):
        print(f'H{i} = ', calc_entropy(text, i))
    return l_average

def shannon_encode(line: str) -> float:
    print("\nShannon:\n")

    def decimal_converter(num):
        if num == 0.0:
            return 0.0
        while num > 1:
            num /= 10
        return num

    def float_bin(number: float, places: int):
        whole, dec = str(number).split(".")
        whole = int(whole)
        dec = int(dec)
        res = bin(whole).strip("0b") + "."
        for x in range(places):
            whole, dec = str((decimal_converter(dec)) * 2).split(".")
            dec = int(dec)
            res += whole
        return res

    split_line = list(line[i: i + 1] for i in range(len(line)))

    probabilities = {k: v / len(split_line) for k, v in
Counter(split_line).items()}
    probabilities = dict(sorted(probabilities.items(), key=lambda item: item[1],
reverse=True))

    code_length = [ceil(-log2(i)) for i in probabilities.values()]

    cumulative_probs = [float(0) for _ in range(len(probabilities))]
    for i in range(1, len(probabilities)):
        cumulative_probs[i] = cumulative_probs[i - 1] +
list(probabilities.values())[i - 1]
```

```
    codes = list()
    for i in range(len(cumulative_probs)):
        codes.append(float_bin(cumulative_probs[i], code_length[i])[1:])
        print(f"{list(probabilities.keys())[i]}:
{list(probabilities.values())[i]:.4f} - {codes[i]}")

    l_average = sum(list(probabilities.values())[i] * code_length[i] for i in
range(len(probabilities.items())))
    print("L avg = ", l_average)

    print("coded_shannon.txt")
    with open("./output/coded_shannon.txt", "w") as f:
        for i in line:
            index = list(probabilities.keys()).index(i)
            f.write(codes[index])

    with open("./output/coded_shannon.txt", "r") as f:
        text = f.readline()
    for i in range(1, 4):
        print(f'H{i} = ', calc_entropy(text, i))
    return l_average

def main():

    print("\n----- Equal probs -----")
    input_text = preprocess_file('./input/equal_prob.txt', 'en')

    orig_entropy = calc_entropy(input_text, 1)
    print(f"H = {orig_entropy[0]}\np = {orig_entropy[1]}")

    l_avg_huf = huffman_encode(input_text)
    l_avg_shan = shannon_encode(input_text)
    print(f"\nr huffman = {l_avg_huf - orig_entropy[0]}\nr shannon = {l_avg_shan -
orig_entropy[0]}")

    print("\n----- Different probs -----")
    input_text = preprocess_file('./input/diff_prob.txt', 'en')

    orig_entropy = calc_entropy(input_text, 1)
    print(f"H = {orig_entropy[0]}\np = {orig_entropy[1]}")

    l_avg_huf = huffman_encode(input_text)
    l_avg_shan = shannon_encode(input_text)
    print(f"\nr huffman = {l_avg_huf - orig_entropy[0]}\nr shannon = {l_avg_shan -
orig_entropy[0]}")

    print("\n----- text.txt -----")
    input_text = preprocess_file('./input/text.txt', 'ru')
    orig_entropy = calc_entropy(input_text, 1)
    print(f"H = {orig_entropy[0]}\np = {orig_entropy[1]}")
```

```
    l_avg_huf = huffman_encode(input_text)
    l_avg_shan = shannon_encode(input_text)
    print(f"\nr huffman = {l_avg_huf - orig_entropy[0]}\nr shannon = {l_avg_shan -
orig_entropy[0]}")

if __name__ == "__main__":
    main()
```

```
    split_line = list(line[i: i + 1] for i in range(len(line)))

    probabilities = {k: v / len(split_line) for k, v in
Counter(split_line).items()}
    probabilities = dict(
        sorted(probabilities.items(), key=lambda item: item[1],
reverse=True)
    )
    # print(probabilities)

    code_length = [ceil(-log2(i)) for i in probabilities.values()]
    # print(code_length)

    cumulative_probs = [float(0) for _ in range(len(probabilities))]
    for i in range(1, len(probabilities)):
        cumulative_probs[i] = (
            cumulative_probs[i - 1] + list(probabilities.values())[i -
1]
        )
    # print(cumulative_probs)

    codes = list()
    for i in range(len(cumulative_probs)):
        codes.append(float_bin(cumulative_probs[i],
code_length[i])[1:])
        print(
            f"{list(probabilities.keys())[i]}:
{list(probabilities.values())[i]:.4f} - {codes[i]}"
        )

    # print(codes)
    l_average = sum(
        list(probabilities.values())[i] * code_length[i]
        for i in range(len(probabilities.items()))
    )
    print("L avg = ", l_average)

    print("coded_shannon.txt")
```

```
    with open("./output/coded_shannon.txt", "w") as f:
        for i in line:
            index = list(probabilities.keys()).index(i)
            f.write(codes[index])

    with open("./output/coded_shannon.txt", "r") as f:
        text = f.readline()
    for i in range(1, 4):
        print(f'H{i} = ', calc_entropy(text, i))
    return l_average

def main():
    print("\n~~~text.txt~~~")
    input_text = preprocess_file('./input/text.txt', 'ru')
    orig_entropy = calc_entropy(input_text, 1)
    print(f"H = {orig_entropy[0]}\np = {orig_entropy[1]}")

    l_avg_huf = huffman_encode(input_text)
    l_avg_shan = shannon_encode(input_text)
    print(f"\nr huffman = {round(l_avg_huf - orig_entropy[0], 5)}\nr
shannon = {round(l_avg_shan - orig_entropy[0], 5)}")


if __name__ == "__main__":
    main()
```