МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчет по лабораторной работе №9**

**СЖАТИЕ/РАСПАКОВКА ДАННЫХ НА ОСНОВЕ СТАТИСТИЧЕСКИХ МЕТОДОВ**

Выполнил:

Cтудент 3 курса 1 группы

Парибок И. А.

Вариант 5

Минск 2022

**Цель**: приобретение практических навыков использования  
статистических методов Шеннона − Фано и Хаффмана (Shannon-Fano and Huffman coding) для сжатия/распаковки данных.

**Практическое задание:**

1. Разработать авторское приложение в соответствии с целью  
лабораторной работы.   
 2. С помощью приложения выполнить прямое и обратное  
преобразования сообщения, состоящего из собственных имени и  
фамилии.   
 Сжатие/распаковка данных на основе статистических методов 93  
Можно использовать любой из известных методов сортировки  
символов массива. Метод кодировки (Шеннона−Фано, Хаффмана) использовать по указанию преподавателя.   
 При этом таблица отсортированных символов строится:  
 а) на основе данных, полученных в лабораторной работе №2;   
 б) динамически, на основе анализа сжимаемого сообщения.   
 3. Определить эффективность (в сравнении с кодами ASCII) сжатия сообщения.   
 4. Результаты оформить в виде отчета по установленным  
правилам.

**Выполнение работы:**

Реализация алгоритма Шеннона−Фано представлено в Приложении А. А реализация алгоритма Хаффмана представлено в Приложении B.



Рис 1 – Исходное сообщение

В первом случае алфавит строится на основе исходного сообщения, частота использований символов алфавита представлена на рисунке 2, исходя из частотных характеристик алгоритм Хаффмана и алгоритм Шеннона − Фано наиболее часто встречающемся символам выделяют более короткие кодирующие кода. Символы алфавита и их бинарное представление изложены в рисунке 2.

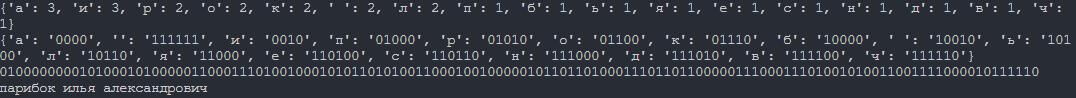


Рис 2 – Частота использование символов алфавита. Бинарное представление символов алфавита. Кодирование/декодирование исходного сообщения методом Шеннона-Фано.

Во втором случае используется русский алфавит. Его статистическая частотность представлена на рисунке 3. Выполнение алгоритма Хаффмана на основе русского алфавита показано на рисунке 4.

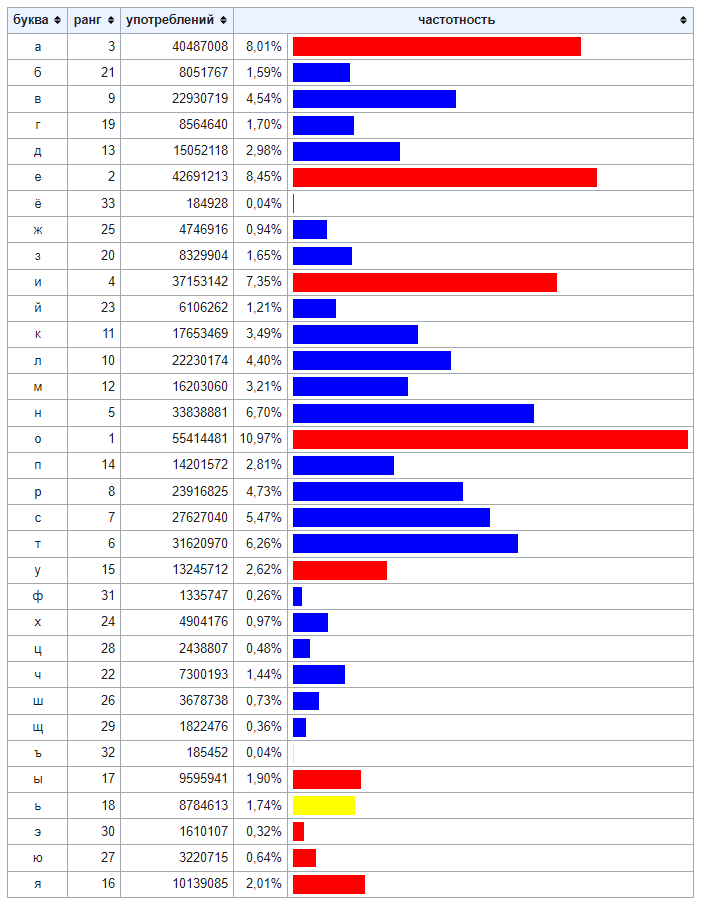


Рис 3 – Статистика частотности букв русского языка (на материале НКРЯ)

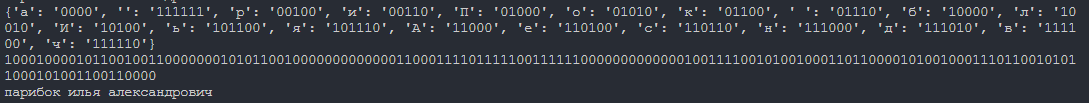


Рис 4 – Бинарное представление символов русского алфавита. Кодирование/декодирование исходного сообщения методом Шеннона-Фано



Рис 5 – Разница эффективности сжатий

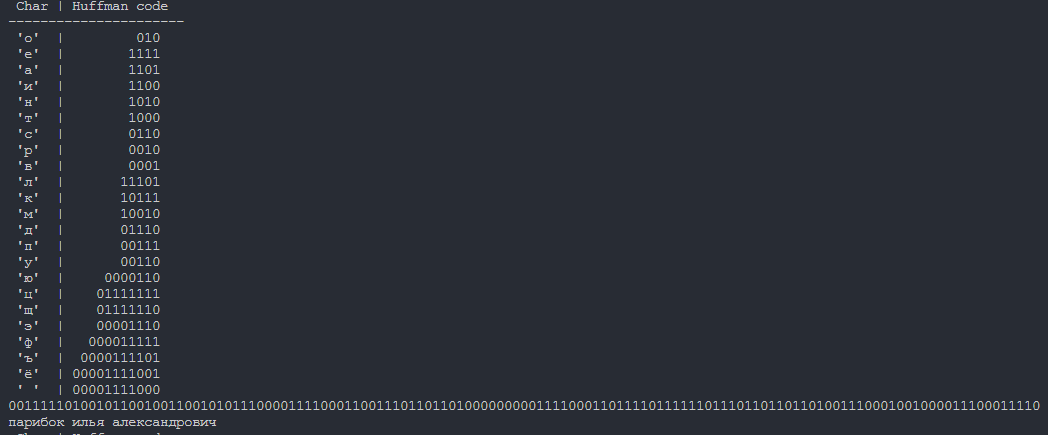


Рис 6 – Бинарное представление символов русского алфавита. Кодирование/декодирование исходного сообщения алгоритмом Хаффмана

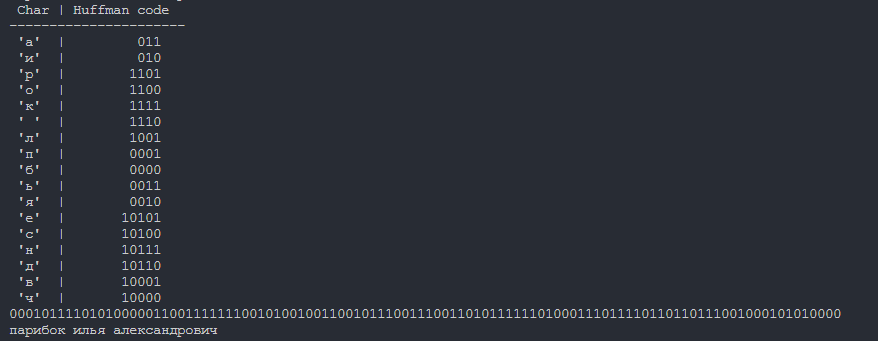


Рис 7 – Бинарное представление символов русского алфавита. Кодирование/декодирование исходного сообщения алгоритмом Хаффмана



Рис 8 – Разница эффективности сжатий

**Вывод**: в результате данной лабораторной работы было разработано приложение для с использованием методов Шеннона − Фано и Хаффмана для сжатия/распаковки данных. В ходе сравнения, было выявлено что для изложенного в рисунке 1 исходного сообщения, динамические алфавиты эффективней, и что метод методов Шеннона − Фано и Хаффмана сократили длину сообщения в двоичном виде в сравнение с кодами ASCII.

Приложение А

def build\_shannon\_fano\_tree(symbols, frequencies):

    if len(symbols) == 1:

        return (symbols[0], '')

    else:

        frequencies = list(frequencies)

        total\_frequency = sum(frequencies)

        half\_frequency = total\_frequency / 2

        cumulative\_frequency = 0

        left\_side = []

        left\_frequencies = []

        right\_side = []

        right\_frequencies = []

        for i in range(len(symbols)):

            symbol = symbols[i]

            frequency = frequencies[i]

            if cumulative\_frequency + frequency <= half\_frequency:

                left\_side.append(symbol)

                left\_frequencies.append(frequency)

                cumulative\_frequency += frequency

            else:

                right\_side.append(symbol)

                right\_frequencies.append(frequency)

        left\_branch = build\_shannon\_fano\_tree(left\_side, left\_frequencies)

        right\_branch = build\_shannon\_fano\_tree(right\_side, right\_frequencies)

        return (left\_branch, right\_branch)

def build\_shannon\_fano\_code(tree, prefix=''):

    if isinstance(tree, tuple):

        left\_branch, right\_branch = tree

        left\_code = build\_shannon\_fano\_code(left\_branch, prefix + '0')

        right\_code = build\_shannon\_fano\_code(right\_branch, prefix + '1')

        return {\*\*left\_code, \*\*right\_code}

    else:

        return {tree: prefix}

def shannon\_fano(symbols, frequencies):

    tree = build\_shannon\_fano\_tree(symbols, frequencies)

    code = build\_shannon\_fano\_code(tree)

    return code

def get\_char\_frequency(s):

*# создаем словарь, где каждому символу ставим в соответствие его частоту*

    char\_frequency = {}

    for c in s:

        if c in char\_frequency:

            char\_frequency[c] += 1

        else:

            char\_frequency[c] = 1

    return char\_frequency

def decode\_shannon\_fano(code, encoded\_message):

    decoded\_message = ''

    while encoded\_message:

        for symbol, symbol\_code in code.items():

            if encoded\_message.startswith(symbol\_code):

                decoded\_message += symbol

                encoded\_message = encoded\_message[len(symbol\_code):]

                break

    return decoded\_message

def get\_percentage\_difference(length1, length2):

    difference = abs(length1 - length2)

    max\_length = max(length1, length2)

    percentage\_difference = difference / max\_length \* 100

    return percentage\_difference

*# используем встроенную функцию Counter из модуля collections*

FIO = "парибок илья александрович"

sorted\_fio = dict(sorted(get\_char\_frequency(FIO).items(),

                  key=lambda x: x[1], reverse=True))

print(sorted\_fio)

symbols = list(sorted\_fio.keys())

frequencies = list(sorted\_fio.values())

code = shannon\_fano(symbols, frequencies)

print(code)

coded\_fio = ""

for i in FIO:

    coded\_fio += code[i]

print(coded\_fio)

decoded\_fio = decode\_shannon\_fano(code, coded\_fio)

print(decoded\_fio)

cirylic = {" ": 228, "а": 40487008, "б": 8051767, "в": 22930719, "г": 8564640, "д": 15052118, "е": 42691213, "ё": 184928, "ж": 4746916, "з": 8329904, "и": 37153142, "й": 6106262, "к": 17653469, "л": 22230174, "м": 16203060, "н": 33838881, "о": 55414481,

           "п": 14201572, "р": 23916825, "с": 27627040, "т": 31620970, "у": 13245712, "ф": 1335747, "х": 4904176, "ц": 2438807, "ч": 7300193, "ш": 3678738, "щ": 1822476, "ъ": 185452, "ы": 9595941, "ь": 8784613, "э": 1610107, "ю": 3220715, "я": 10139085}

symbols\_cirylic = list(cirylic.keys())

frequencies\_cirylic = list(cirylic.values())

code\_cirylic = shannon\_fano(symbols\_cirylic, frequencies\_cirylic)

print(code)

coded\_fio\_cirylic = ""

for i in FIO.lower():

    coded\_fio\_cirylic += code\_cirylic[i]

print(coded\_fio\_cirylic)

decoded\_fio\_irylic = decode\_shannon\_fano(code\_cirylic, coded\_fio\_cirylic)

print(decoded\_fio\_irylic)

print("Разница между динамическим и статическим русским алфавитом: " +

      str(get\_percentage\_difference(len(coded\_fio\_cirylic), len(coded\_fio))) + " %")

print("Разница между Шеннона и ASCII: " +

      str(get\_percentage\_difference(len(FIO) \* 8, len(coded\_fio))) + " %")

*# Huffman Coding in python*

def decode\_shannon\_fano(code, encoded\_message):

    decoded\_message = ''

    while encoded\_message:

        for symbol, symbol\_code in code.items():

            if encoded\_message.startswith(symbol\_code):

                decoded\_message += symbol

                encoded\_message = encoded\_message[len(symbol\_code):]

                break

    return decoded\_message

def get\_percentage\_difference(length1, length2):

    difference = abs(length1 - length2)

    max\_length = max(length1, length2)

    percentage\_difference = difference / max\_length \* 100

    return percentage\_difference

*# используем встроенную функцию Counter из модуля collections*

string = 'парибок илья александрович'

*# Creating tree nodes*

class NodeTree(object):

    def \_\_init\_\_(self, left=None, right=None):

        self.left = left

        self.right = right

    def children(self):

        return (self.left, self.right)

    def nodes(self):

        return (self.left, self.right)

    def \_\_str\_\_(self):

        return '%s\_%s' % (self.left, self.right)

*# Main function implementing huffman coding*

def huffman\_code\_tree(node, left=True, binString=''):

    if type(node) is str:

        return {node: binString}

    (l, r) = node.children()

    d = dict()

    d.update(huffman\_code\_tree(l, True, binString + '0'))

    d.update(huffman\_code\_tree(r, False, binString + '1'))

    return d

freq =  {" ": 228,"а":40487008,"б":8051767,"в":22930719,"г":8564640,"д":15052118,"е":42691213,"ё":184928,"ж":4746916,"з":8329904,"и":37153142,"й":6106262,"к":17653469,"л":22230174,"м":16203060,"н":33838881,"о":55414481,"п":14201572,"р":23916825,"с":27627040,"т":31620970,"у":13245712,"ф":1335747,"х":4904176,"ц":2438807,"ч":7300193,"ш":3678738,"щ":1822476,"ъ":185452,"ы":9595941,"ь":8784613,"э":1610107,"ю":3220715,"я":10139085}

freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

nodes = freq

*# nodes =  [("а":40487008),("б":8051767),("в":22930719),("г":8564640),("д":15052118),("е":42691213),("ё":184928),("ж":4746916),("з":8329904),("и":37153142),("й":6106262),("к":17653469),("л":22230174),("м":16203060),("н":33838881),("о":55414481),("п":14201572),("р":23916825),("с":27627040),("т":31620970),("у":13245712),("ф":1335747),("х":4904176),("ц":2438807),("ч":7300193),("ш":3678738),("щ":1822476),("ъ":185452),("ы":9595941),("ь":8784613),("э":1610107),("ю":3220715),("я":10139085)]*

while len(nodes) > 1:

    (key1, c1) = nodes[-1]

    (key2, c2) = nodes[-2]

    nodes = nodes[:-2]

    node = NodeTree(key1, key2)

    nodes.append((node, c1 + c2))

    nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffman\_code\_tree(nodes[0][0])

print(' Char | Huffman code ')

print('----------------------')

for (char, frequency) in freq:

    print(' %-4r |%12s' % (char, huffmanCode[char]))

coded\_fio\_cirylic = ""

for i in string:

    coded\_fio\_cirylic += huffmanCode[i]

print(coded\_fio\_cirylic)

decoded\_fio\_cyrilic = decode\_shannon\_fano(huffmanCode, coded\_fio\_cirylic)

print(decoded\_fio\_cyrilic)

*# # Calculating frequency*

freq = {}

for c in string:

    if c in freq:

        freq[c] += 1

    else:

        freq[c] = 1

freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

nodes = freq

while len(nodes) > 1:

    (key1, c1) = nodes[-1]

    (key2, c2) = nodes[-2]

    nodes = nodes[:-2]

    node = NodeTree(key1, key2)

    nodes.append((node, c1 + c2))

    nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffman\_code\_tree(nodes[0][0])

print(' Char | Huffman code ')

print('----------------------')

for (char, frequency) in freq:

    print(' %-4r |%12s' % (char, huffmanCode[char]))

coded\_fio = ""

for i in string:

    coded\_fio += huffmanCode[i]

print(coded\_fio)

decoded\_fio = decode\_shannon\_fano(huffmanCode, coded\_fio)

print(decoded\_fio)

print("Разница длин сообщений: " + str(get\_percentage\_difference(len(coded\_fio\_cirylic), len(coded\_fio))) + " %")

print("Разница между Хаффмана и ASCII: " + str(get\_percentage\_difference(len(string)\* 8, len(coded\_fio))) + " %")