Лабораторная №3 “Методы кодирования”

Вариант №30

1) Код Хемминга

Смотреть Excel-файл: решение задания 1

2) Расстояние Хемминга

Двоичные коды с расстоянием не менее 2:

н	00000	с	10010
о	01001	т	10100
п	10001	ф	00110
р	01100	ч	00011

Программный код:

```
from tabulate import tabulate
m = ["", 'н', 'о', 'п', 'р', 'с', 'т', 'ф', 'х']
n = ['00000', '01001','10001','01100','10010','10100','00110','00011']
table = []
for i in range(0, len(n)):
        line = [m[i + 1]]
        for l in range(0, i):
                line.append("")
        line.append("-")
        for j in range(i+1, len(n)):
                num = 0
                for k in range(0, len(n[i])):
                        if n[i][k] != n[j][k]:
                                num += 1
                line.append(str(num))
        table.append(line)
print(tabulate(table, m, tablefmt="psql",numalign='center'))
```

Таблица попарных расстояний представлена в программах.

Декодер исправляет все ошибки, кратность которых не превышает

$$l \leq INT\left[\frac{d_{min} - 1}{2}\right]$$

Минимальное расстояние d(min)=2 => код может исправить ноль ошибок, обнаружить одну ошибку

Пусть было получено сообщение 00100. Совпадений кода с кодами в таблице двоичных кодов нет.

Расстояния Хемминга для полученного кода и исходного набора букв:

н: d(00000,00100)=1

о: d(01001,00100)=3

п: d(10001,00100)=1

р: d(01100,00100)=1

с: d(10010,00100)=3

т: d(10100,00100)=1

ф: d(00110,00100)=1,

х: d(00011,00100)=3
Минимальное расстояние d(min)=1 соответствует пяти символам. Найдена одиночная ошибка(это ситуация, когда в процессе передачи одного кодового слова происходит изменение одного бита), исправление невозможно. (Если при передаче кодового слова по каналу связи в нем произошла одиночная ошибка, то расстояние Хемминга между переданным словом и принятым вектором будет равно единице. Если при этом одно кодовое слово не перешло в другое, то ошибка будет обнаружена при декодировании.)

Двоичные коды с расстоянием не менее 3:

н	00000000	с	11000001
о	00000111	т	01010101
п	00011001	ф	10110010
р	10100100	х	11111111

Программный код:

```
from tabulate import tabulate
m = ["", 'н', 'о', 'п', 'р', 'с', 'т', 'ф', 'х']
n = ['00000000', '00000111', '00011001', '11000001', '01010101', '10110010', '10100100',
'11111111']
table = []
for i in range(0, len(n)):
        line = [m[i + 1]]
        for l in range(0, i):
                line.append("")
        line.append("-")
        for j in range(i+1, len(n)):
                num = 0
                for k in range(0, len(n[i])):
                        if n[i][k] != n[j][k]:
                                num += 1
                line.append(str(num))
        table.append(line)
print(tabulate(table, m, tablefmt="psql",numalign='center'))
```

Таблица попарных расстояний представлена в программе.

Минимальное расстояние d(min)=3, значит такой код может исправить одиночную ошибку, обнаружить две ошибки
Пусть было получено сообщение 00001110. Совпадений с кодами в таблице двоичных кодов не обнаружено. Посчитаем расстояния Хемминга для полученного кода и исходного набора букв:
н: d(00000000,00001110)=3

о: d(00000111,00001110)=2
п: d(00011001,00001110)=4
р: d(11000001,00001110)=6
с: d(01010101,00001110)=5
т: d(10110010,00001110)=5
ф: d(10100100,00001110)=4
х: d(11111111,00001110)=5
Минимальное расстояние d(min)=2 соответствует символу «о». Найдена и исправлена одиночная ошибка

3) Сжатие способом кодирования серий

Программный код:

```
from itertools import groupby
from collections import Counter
import re
string = "aaaaadgggggggggggggghtyikloooooop"
print(string)
amount = 1
final_string = ""
for i in range(len(string)-1):
    if string[i]==string[i+1]:
        amount += 1
    else:
        a = string[i]
        final_string = final_string +str(amount)+string[i]+" "
        amount  = 1
if string[len(string)-1]!= string[len(string)-2]:
    final_string = final_string + str(amount) + string[len(string)-1]+" "
    print(final_string)
    final_string = final_string.split(" ")
    final_string.pop(len(final_string)-1)
    final_string1 = final_string[:]
    for i  in range(0,len(final_string)):
        final_string[i]=re.sub("[^0-9]","",final_string[i])
        final_string1[i]=re.sub("[^A-Za-z]","",final_string1[i])
    for i in range(0,len(final_string)-1):
        j = i+1
        c=0
        while(final_string[i]==final_string[j]):
            a=final_string1[i]
            b=final_string1[j]
            final_string1[i]=a+b
            j += 1
            c+=1
        for p in range(1,c+1):
            final_string[i+p]=" "
            final_string1[i+p]=" "
    for i in range(0,len(final_string)-1):
        if(final_string1[i].find(" ")!= -1):
```

```
        final_string1[i]=" "
final_string1=[el for el, _ in groupby(final_string1)]
c= Counter(final_string1)
for _ in range(0, c[' ']):
    final_string1.remove(" ")
final_string=[el for el, _ in groupby(final_string)]
c= Counter(final_string)
for _ in range(0, c[' ']):
    final_string.remove(" ")
A=[""]*len(final_string)
for i in range(0, len(final_string)):
    A[i]=final_string[i]+final_string1[i]
print(*A)
```

```
aaaaadgggggggggggggghtyiklooooop
5a 1d 15g 1h 1t 1y 1i 1k 1l 5o 1p
5a 1d 15g 1htyikl 5o 1p
>>>
```

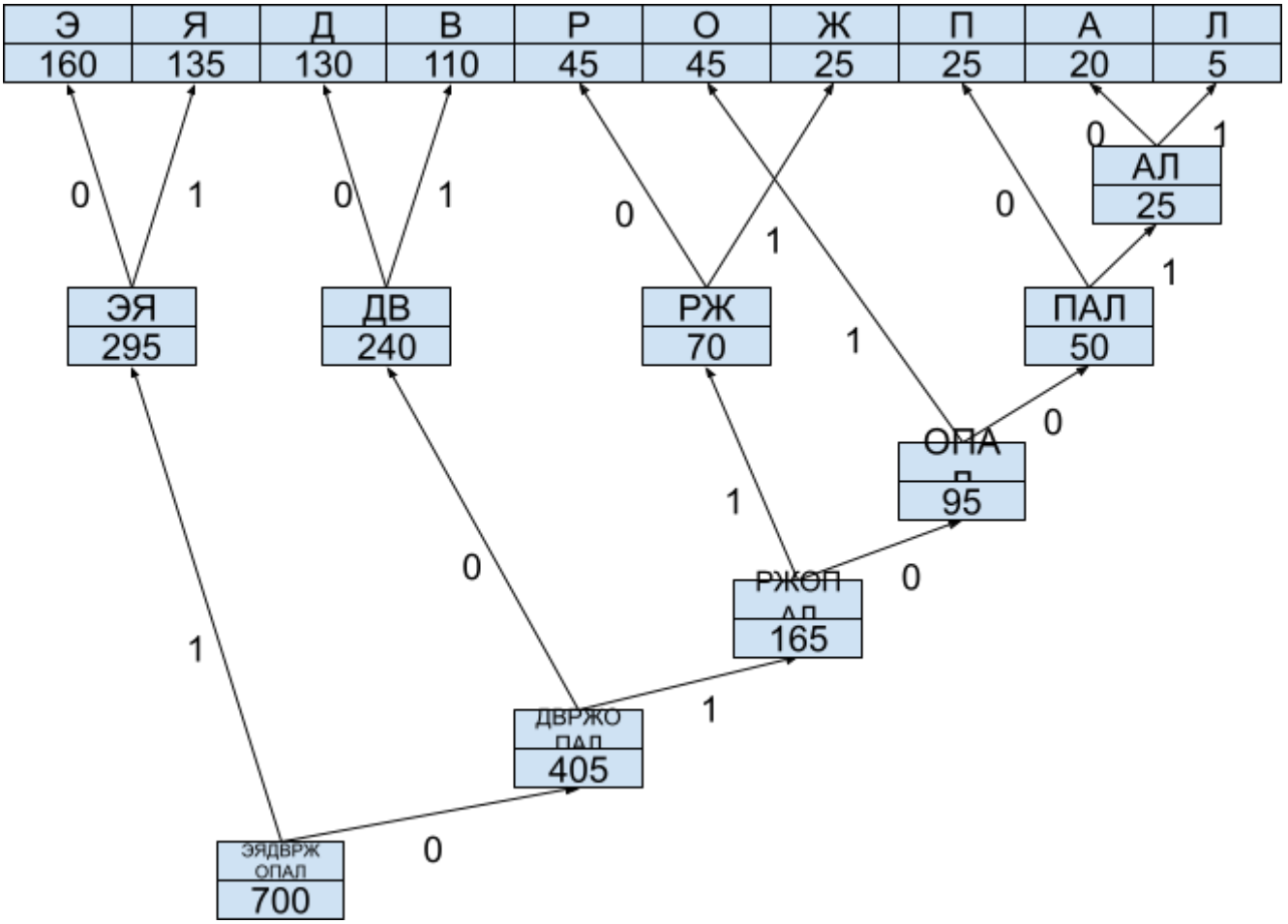
Строка “5a1d15g1h1t1y1i1k1l5o1p” содержит 23 символа. Изначальная строка - 33 символа. Одиннадцать символов a,d,g,h,t,y,i,k,l,o,p можно закодировать следующим образом:

a	0000	g	0010	t	0100	i	0110	l	1000	p	1010
d	0001	h	0011	y	0101	k	0111	o	1001		

Тогда строка aaaaadgggggggggggggghtyiklooooop будет состоять из 33·4=132 символов.
Степень сжатия: 100-23/132·100 ≈ 82,58%

4)Алгоритм Хаффмана

В	А	П	Р	О	Л	Д	Ж	Э	Я
110	20	25	45	45	5	130	25	160	135



В	001
А	010010
П	01000
Р	0110
О	0101
Л	010011
Д	000
Ж	0111
Э	10
Я	11

Например, закодируем слово ЖДЭЯД с помощью таблицы (01110001011000, 14 символов). Раскодируем, двигаясь по дереву.

При равномерном кодировании каждый символ кодируется $10 = 2^i => i = 4$ символами

Степень сжатия по сравнению с равномерным: $100 - \frac{14}{5*4} * 100 = 30\%$

5)Арифм.кодирование:
Арифметическое кодирование — один из алгоритмов энтропийного сжатия.
Является альтернативой методу сжатия по Хаффману. Код присваивается всему передаваемому файлу, вместо кодирования отдельных символов, что дает большую гибкость в представлении дробных частот встречаемости символов.
Кодер читает входной файл символ за символом и добавляет биты к сжатому файлу.

Получающийся код представляет собой дробную часть числа из полуинтервала [0, 1).
Рассмотрим на координатной прямой отрезок от 0 до 1. Назовём этот отрезок рабочим.
Расположим на нем точки таким образом, что длины образованных отрезков будут равны частоте использования символа, и каждый такой отрезок будет соответствовать одному символу. Теперь возьмём символ из потока и найдём для него отрезок среди только что сформированных, теперь отрезок для этого символа стал рабочим.
Разобьем его так же, как разбили отрезок от 0 до 1.
Выполним эту операцию для некоторого числа последовательных символов. Затем выберем любое число из рабочего отрезка.

Биты этого числа вместе с длиной его битовой записи и есть результат арифметического кодирования использованных символов потока.

Программный код:

```
def float2bin(x, eps=1e-9):
	res = ''
	while x > eps:
		x *= 2
		res += str(int(x))
		x -= int(x)
	return res


def bin2float(x):
	return sum(2 ** (-i - 1) for i, digit in enumerate(x) if digit == '1')


def find_code(a, b):
	i = 0
	a += '0' * (len(b) - len(a))
	while a[i] == b[i]:
		i += 1
	res = a[:i] + '0'
	cnt = 0
	while a[i] == 1:
		i += 1
		cnt += 1
	res += '1' * (cnt + 1)
	return res


def coding(word, alphabet, p):
	left, right = 0, 1
	for letter in word:
		i = alphabet.index(letter)
		left, right = (left + (right - left) * sum(p[:i]), left + (right - left) * sum(p[: i + 1]))
	return find_code(*map(float2bin, (left, right)))


def decoding(code, length, alphabet, p):
	code = bin2float(code)
```

```
    word = ''
    left, right = 0, 1
    for _ in range(length):
        for i, letter in enumerate(alphabet):
            interval = (left + (right - left) * sum(p[:i]), left + (right - left) * sum(p[:i + 1]))
            if interval[0] <= code < interval[1]:
                word += letter
                code = (code - interval[0]) / (interval[1] - interval[0])
                break
    return word


alphabet = 'abcdef'
p = (0.05, 0.10, 0.05, 0.55, 0.15, 0.10)
word = 'eacdbf'
code = coding(word, alphabet, p)
print(code)
print(decoding(code, len(word), alphabet, p))
```

```
11000000010100001001
eacdbf
>>>
```

Строка 11000000010100001001 содержит 20 символов. Строка eacdbf содержит 6 символов. Шесть символов a,b,c,d,e,f можно закодировать так:

a	000	c	010	e	100
b	001	d	011	f	101

Тогда строка eacdbf будет состоять из 6·3=18 символов.

Степень сжатия: $100 - \frac{20}{18} * 100 \approx -11,11\%$