МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

ЛАБОРАТОРНАЯ РАБОТА

Алгоритм Фано

студента 4 курса 431 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Серебрякова Алексея Владимировича

Научный руководитель

доцент, к. п. н. _____ А. С. Гераськин

подпись, дата

Саратов 2022

Код Шеннона

Код Шеннона — алгоритм префиксного кодирования алфавита, предложенный Клодом Шенноном, в котором используется избыточность сообщения, заключённая в неоднородном распределении частот символов первичного алфавита, то есть заменяет коды более частых символов короткими последовательностями, а коды более редких символов — более длинными последовательностями.

Определение:

Пусть $A = \{a_1, a_2, \ldots, a_n\}$ — алфавит из n различных символов, которому соответствует набор вероятностей $P = \{p_1, p_2, \ldots, p_n\}$ такой, что $p_x \geqslant p_y, x < y$.

$b_x = \sum\limits_{i \in [1, x-1]} p_i$. Тогда набор бинарных кодов $C = \{c_1, c_2, \ldots, c_n\}$, такой, что:

1. c_i не является префиксом для c_j, при $i \neq j$
2. c_i представляет собой $\lceil -\log p_i \rceil$ коэффициентов двоичного разложения числа b_i

называется **кодом Шеннона**.

Алгоритм построения бинарного кода Шеннона

Пусть нам даны наборы A и P, тогда для нахождения кодовых слов необходимо:

1. Отсортировать элементы алфавита по не возрастанию вероятности встречи символа.
2. Элементу a_x поставить в соответствие число $b_x = \sum\limits_{i \in [1, x-1]} p_i$, при этом $b_1 = 0$.
3. Представить каждое число b_x в виде двоичной дроби.
4. В качестве кодового слова для a_x использовать первые $L_x = \lceil -\log p_x \rceil$ коэффициентов представления b_x. ($\lceil z \rceil$ — наименьшее целое число, не меньшее z)

Пример

Для примера возьмём алфавит $A = \{a, b, c, d, e, f\}$ и набор P:

Символ	a	b	c	d	e	f
p_x	0.10	0.20	0.10	0.10	0.35	0.15

По алгоритму сортируем элементы алфавита по не возрастанию p_x:

Символ	e	b	f	a	c	d
p_x	0.35	0.20	0.15	0.10	0.10	0.10

Каждому символу a_x сопоставляем b_x:

Символ	e	b	f	a	c	d
b_x	0.00	0.35	0.55	0.70	0.80	0.90

Переведём b_x в двоичную систему счисления:

Символ	e	b	f	a	c	d
b_x	0.00000	0.01010	0.10001	0.10110	0.11001	0.11100

Посчитаем L_x и запишем коды:

Символ	e	b	f	a	c	d
L_x	2	3	3	4	4	4
Код	00	010	100	1011	1100	1110

Код программы

```
import time
import math

ans = int(input('Сжать - 0, Разжать - 1\n'))


def to_bin(x):
    s = ''
    for i in range(30):
        s += str(int((x*2)//1))
        x = (x*2) % 1
    return s


if ans == 0:

    a = open('studies\\tkisi\\Тест_8.txt')
    start_time = time.perf_counter()

    text = a.read()
    input_text = text

    d = []
    alph = []

    # Пробежимся по строке и будем добавлять в список пары (частота, символ) а затем удалять все вхождения символа из строки
    while len(text) > 0:
        x = text[0]
        alph.append(x)
        d.append((text.count(x), x))
        text = text.replace(x, '')

    # Сортировка списка будет моделировать очередь с приоритетом
    d.sort()

    print('Алфавит:', alph)
    print('Частота встречаемости', d)
    d.reverse()
    d_com = {}

    lens = len(input_text)
    d_p = {}
    for x in d:
        d_p[x[1]] = x[0] / lens

    d_com[d[0][1]] = 0.0
    iters = 0.0
    iters = d[0][0] / lens
    for x in d[1:]:
        d_com[x[1]] = iters
        iters += x[0] / lens

    d_log = {}
    for x in d:
        d_log[x[1]] = math.ceil(-math.log2(d_p[x[1]]))

    print(d_p)
```

```
    print(d_com)
    print(d_log)

    dictionary = {}
    g = 0
    for x in d:
        if g == 0:
            dictionary[x[1]] = '1' + '0'*d_log[x[1]]
            g = 1
        else:
            dictionary[x[1]] = '0' + to_bin(d_com[x[1]])[:d_log[x[1]]]

    print(dictionary)

    print('Сопоставим исходным символам их код: ', dictionary)

    bin_file = open('res3.bin', 'wb+')

    l = list(dictionary.keys())
    bin_file.write(b' ')
    for x in l:
        s2 = int(dictionary[x], 2)
        bin_file.write(x.encode())
        bin_file.write(dictionary[x].encode())
        bin_file.write(b' ')

    if (l.count('\n') == 0):
        bin_file.write(b'\n')
    s = ''
    bin_file.write(b'\n')
    for x in input_text:
        s += dictionary[x]
    #print('\n', s, len(s))

    if len(s) % 8 != 0:
        bin_file.write(str((8*(len(s)//8) + 8 - len(s))).encode())
        s = '0' * (8*(len(s)//8) + 8 - len(s)) + s
    else:
        bin_file.write((str(0)).encode())
    bin_file.write(b'\n')

    #print('\n', s, len(s))

    while s != '':
        sub = s[0:8]
        s = s[8:]
        s1 = int(sub, 2)
        s2 = s1.to_bytes((s1.bit_length() + 7) // 8, 'big')
        bin_file.write(s1.to_bytes((s1.bit_length() + 7) // 8, 'big'))
    bin_file.close()

    print("--- %s seconds ---" % (time.perf_counter() - start_time))


else:

    bf = open('res3.bin', 'rb')

    start_time = time.perf_counter()
```

```
decode_text = bf.readline().decode('utf-8')
decode_text += bf.readline().decode('utf-8')
decode_text += bf.readline().decode('utf-8')
# print(decode_text)

output_text = bf.read()
a = ''
for x in reversed(output_text):
    a = '0' * (8 - len(bin(x)[2:])) + bin(x)[2:] + a

if len(a) % 8 != 0:
    a = '0' * (8*(len(a)//8) + 8 - len(a)) + a

output_text = decode_text + a

# print(output_text)

dec_dict = {}

t = 0
a = ''
t2 = 0
for x in output_text:
    if x == '\n' and t == 1:
        break
    if x == '\n':
        t += 1
    if x != ' ':
        a += x
        if t2 != 0:
            t2 = 0
    if x == ' ' and t2 == 1:
        t2 = -1
        a += x
    if x == ' ' and t2 != -1 and a != '':
        t2 = 1
        dec_dict[a[1:]] = a[0]
        a = ''
    if x == ' ' and t2 != -1 and a == '':
        t2 = 1

output_text = output_text[output_text.index('\n')+1:]
output_text = output_text[output_text.index('\n')+1:]

zero = int(output_text[0])
if zero != 7:
    output_text = output_text[2+zero:]
else:
    output_text = '0' + output_text[2:]
#print(dec_dict, output_text)

out_f = open('restore3.txt', 'w')

final_text = ''
a = ''
l = list(dec_dict.keys())
for x in output_text:
    a += x
    if l.count(a) != 0:
        out_f.write(dec_dict[a])
        final_text += dec_dict[a]
```

```
        a = ''

# out_f.write(final_text)
out_f.close()
print("--- %s seconds ---" % (time.perf_counter() - start_time))
```