Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное

учреждения высшего образования

«Национальный исследовательский технологический университет «МИСиС»

Лабораторная работа №3 по дисциплине

«Основы теории информации» на тему:

«Основная задача кодирования»

Направление подготовки:

09.03.01 «Информатика и вычислительная техника» Семестр 5

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| Попов Егор  Борисович  (Ф.И.О. студента) | Куприянов Вячеслав Васильевич  (Ф.И.О преподавателя) |
| БИВТ-22-СП-4  (№ группы) | (оценка) |
| (дата сдачи) | (дата проверки) |
| Подпись: | Подпись: |

Москва – 2024

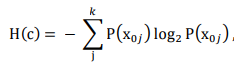
# Цель работы:

Изучить тему «Основная задача кодирования» и реализовать решение задачи №3.1 (стр. 12) при помощи языка программирования Python 3.10 для закрепления материала.

1. **Теоретическая часть**

Средняя длина кодовой комбинации: , где 𝐻(𝜀) – энтропия сообщения, m – количество символов в кодировке.

Энтропия сообщения – количество информации, содержащейся в любом осредненном сообщении. , где



x0 − множество элементов сообщения



Единица измерения энтропии сообщения

Величина избыточности источника: 𝜌𝑢 = 𝑛𝑐− 𝑛𝑐𝑝 𝑚𝑖𝑛 𝑛𝑐 , где 𝑛𝑐 – округленное в большую сторону значение 𝑛𝑐𝑝 𝑚𝑖𝑛 .

Минимальная и необходимая длина кодовой комбинации: 𝑚𝑖𝑛 = 𝑖 ∙ 𝑛𝑐𝑝 𝑚𝑖𝑛 , где I – количество букв элементарного сообщения.

Длина кодовой комбинации при заданной длине элементарного сообщения: 𝑛 = 𝑖 ∙ 𝑙𝑖 , где I – количество букв элементарного сообщения, выбранная длина элементарного сообщения.

1. **Условие задачи**

Имеется сообщение ζ=(ζ1,ζ2,ζ3,ζ4,ζ5)*ζ*=(*ζ*1​,*ζ*2​,*ζ*3​,*ζ*4​,*ζ*5​) со следующими значениями вероятности их появления:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ζ | **1** | **2** | **3** | **4** | **5** |
| **P(**ζ**)** | **1/2** | **1/8** | **1/8** | **1/8** | **1/8** |

Сообщение кодируется с помощью алфавита A=(0,1)*A*=(0,1), где m=2*m*=2. Каждое

элементарное сообщение записывается тремя буквами (символами): Ri=3*Ri*=3, i=1,2,3,4,5*i*=1,2,3,4,5.

Кодовые слова, соответствующие элементарным сообщениям, имеют вид:

|  |  |
| --- | --- |
| ζ | Кодовое слово |
| 1 | 000 |
| 2 | 001 |
| 3 | 010 |
| 4 | 011 |
| 5 | 100 |

**Задача:** определить минимально необходимую для качественного кодирования среднюю длину кодовой комбинации.

1. **Практическая часть**

Для реализации решения задачи потребуются функции логарифма и округления в большую сторону, которые нужно подключить из библиотеки math командой from math import log и from math import ceil. Функция log принимает 2 аргумента: аргумент логарифма, основание логарифма. Функция ceil принимает на вход 1 параметр — число, которое нужно округлить.

Функция для вычисления длины кодовой комбинации:

def CodeCombination(length, amount):

return length \* amount

Данная функция принимает 2 параметра: длину элементарного сообщения и количество сообщений, и возвращает их произведение.

Функция для вычисления средней длины кодовой комбинации:

def FindAverageLength(entropy, countSigns):

return entropy / log(countSigns, 2)

Данная функция принимает 1 параметр — энтропию сообщения, и её возвращает отношение к логарифму с параметром countSigns по основанию 2.

Функция для нахождения избыточности кода:

def RedundancyOfCode(entropy, length, countSigns):

ncp = FindAverageLength(entropy, countSigns)

return round((length - ncp) / length \* 100, 3)

Данная функция принимает 3 параметра: энтропию сообщения, выбранную длину элементарного сообщения и количество знаков в элементарном сообщении, затем запускает функцию FindAverageLength для нахождения средней длины кода. В конце функция возвращает избыточность кода.

Функция для поиска энтропии сообщения:

def EntropyMessage(probabilityArray):

result = 0

for i in range(len(probabilityArray)):

result += probabilityArray[i] \* log(probabilityArray[i], 2)

if result != 0:

result \*= -1

print('Энтропия: ', result)

return result

Данная функция получает 2 аргумента: countMessages, хранящий в себе количество сообщений, и countSigns, хранящий в себе количество символов в алфавите кода для кодирования. Так как события равновероятны, то сразу вычисляем вероятность одного события и записываем результат в переменную p. Далее через цикл суммируем произведение вероятностей на логарифм от вероятности по основанию countSign. После чего умножаем результат на -1 и возвращаем его (в случае, когда result = 0, Python выводит «-0», что не совсем корректно, поэтому я добавил проверку).

Функция для ввода и валидации чисел:

def Validation():

while True:

try:

data = int(input('Введите целое число более 0: '))

if data > 0:

return data

print('Введенное значение не больше 0')

except:

print('Вы ввели не число')

print('Попробуйте еще раз')

Данная функция реализует ввод данных и их проверку на условия:

1. Введенное значение является целым числом;
2. Введенное значение больше 0.

Первое условие проверяется через конструкцию try -- except, которая обрабатывает ошибки, не прекращая выполнения программы. То есть при вводе набора символов, который нельзя преобразовать в целочисленный тип данных, управляющий поток перейдет в блок except и выведет сообщение «Вы ввели не число». А при вводе целого числа не больше 0, программа выведет сообщение «Введенное значение не больше 0».

В конце, если значение не удовлетворило условиям, выводится сообщение «Попробуйте еще раз».

Функция для ввода списка дробей:

def InputProbability():

print('Введите количество сообщений')

count = Validation()

probabilityArray = []

while True:

i = 0

while i < count:

prob = input('Введите вероятность появления сообщения в формате "1/2": ')

try:

first = int(prob.split('/')[0])

second = int(prob.split('/')[1])

probabilityArray.append(first / second)

i += 1

except:

print('Данные введены не верно! Попробуйте еще раз.')

if sum(probabilityArray) == 1:

return probabilityArray

print('Помните. Сумма вероятностей должна быть равна 1!')

Данная функция реализует ввод данных и их проверку на условия:

1. Введенное значение удовлетворяет формату «1/2»;
2. Полученное значение можно представить десятичной дробью;
3. Сумма вероятностей равна 1.

Первые условия проверяются через конструкцию try -- except, которая обрабатывает ошибки, не прекращая выполнения программы. То есть при вводе набора символов, который нельзя преобразовать в десятичную дробь, управляющий поток перейдет в блок except и выведет сообщение «Данные введены не верно! Попробуйте еще раз.». В случае, если сумма вероятностей будет не равна 1, программа выведет сообщение «Помните. Сумма вероятностей должна быть равна 1!».

Основная функция:

countOfSignForEncoding = 2

print('Вычисление качества используемого кода')

array = InputProbability()

print('Введите длину записи закодированного сообщения')

length = Validation()

entropy = EntropyMessage(array)

npc = FindAverageLength(entropy, countOfSignForEncoding)

print('Избыточность используемого кода: ', RedundancyOfCode(entropy, length, countOfSignForEncoding), '%')

print('Минимально необходимая длина кодовой комбинации равна: ', CodeCombination(npc, len(array)))

print('Длина кодовой комбинации при заданной длине равна: ', CodeCombination(length, len(array)))

wait = input("Нажмите Enter для завершения")

if \_\_name\_\_ == '\_\_main\_\_':

Main()

Данная функция реализует в себе основную логику работы программы: получает необходимые значения через функцию Validation и InputProbability, получает энтропию сообщения через функцию EntropyMessage, величину средней длины сообщения FindAverageLength. Также выводит на экран сопровождающие сообщения для пользователя. Переменная countOfSignForEncoding отвечает за количество знаков в кодировке. Так как в задании оно по условию задано равным 2 и в подавляющем большинстве случаев также равно 2, то я решил не делать для неё отдельный ввод, а записать в переменную.

Последняя конструкция не является обязательной и нужна для возможности подключения данного Python файла к другому. Программа начинает работу с этой конструкции.

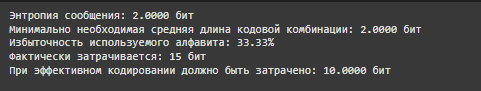
****

Рисунок 1

Пример работы программы на значениях задачи представлен на рисунке 1

**Заключение**

Результаты работы программы показали, что при выбранном способе кодирования затрачивается больше бит, чем необходимо для эффективного кодирования, что свидетельствует о наличии избыточности в используемом коде. Это позволяет сделать вывод о необходимости оптимизации кодирования для повышения эффективности передачи информации.

Таким образом, цель работы была достигнута, и полученные знания и навыки могут быть использованы для решения практических задач в области теории информации и кодирования.

**Приложение**

**Приложение 1**

import math

# Вероятности появления элементарных сообщений

probabilities = [1/2, 1/8, 1/8, 1/8, 1/8]

# Расчет энтропии сообщения

def calculate\_entropy(probabilities):

    entropy = 0

    for p in probabilities:

        entropy -= p \* math.log2(p)

    return entropy

# Расчет минимальной средней длины кодовой комбинации

def calculate\_min\_avg\_length(entropy, m):

    return entropy / math.log2(m)

# Фактическая длина кодовой комбинации

actual\_length = 3

# Расчет энтропии

entropy = calculate\_entropy(probabilities)

print(f"Энтропия сообщения: {entropy:.4f} бит")

# Расчет минимальной средней длины кодовой комбинации

m = 2  # Алфавит A = (0, 1)

min\_avg\_length = calculate\_min\_avg\_length(entropy, m)

print(f"Минимально необходимая средняя длина кодовой комбинации: {min\_avg\_length:.4f} бит")

# Расчет избыточности

redundancy = (actual\_length - min\_avg\_length) / actual\_length \* 100

print(f"Избыточность используемого алфавита: {redundancy:.2f}%")

# Сравнение с фактической длиной

total\_actual\_length = 5 \* actual\_length

print(f"Фактически затрачивается: {total\_actual\_length} бит")

print(f"При эффективном кодировании должно быть затрачено: {5 \* min\_avg\_length:.4f} бит")