Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Российский экономический университет имени Г.В. Плеханова»

**Высшая школа кибертехнологий, математики и статистики**

**Кафедра информатики**

РЕФЕРАТ

По дисциплине “Теоритические основы информатики”

на тему:

Решение задач научной дисциплины “Теоретические основы информатики” посредством использования среды программирования python.

Работу выполнил

студент 1 курса

очной формы обучения

направления: "Информационные системы и технологии”

группы: “15.27Д-ИСТ15/22б”

Лукьянов Андрей Николаевич

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Доцент, Доктор технических наук,

Афанасьева Татьяна Васильевна

Москва, 2022

Содержание:

1. Вступление, пояснение актуальности поставленной темы.
2. Основная часть:
   1. Глава 1: Понятие энтропии, измерение количества информации.
      1. Раздел 1: Информация, энтропия в теории информации.
      2. Раздел 2: нахождение количества информации в сообщении с помощью языка программирования python.
   2. Глава 2: Системы счисления.
      1. Раздел 1: Теория переводов систем счисления. Машинный код.
      2. Раздел 2: Перевод систем счисления, суммирование двоичных чисел, перевод в дополнительный код на языке python.
   3. Глава 3: совершенный дизъюнктивный и конъюнктивный коды на языке python.
3. Заключение

Вступление:

Информация - есть ключевой компонент любой человеческой деятельности. На подобие работы систем физики, химии и прочих естественных наук, информатика представляет из себя метод объяснения и рационализации окружающих человека процессов. Особенность информатики, тогда, заключается в том, что описываемые ею процессы чаще всего антропогенные по своей природе. Это во истину правдиво в отношении информационного процесса. Однако, данный процесс описывает лишь часть природы информации и не перечеркивает неотождествляемость дисциплины от окружающего нас естественного мира. Так, информация присуща любым протекающим в природе процессам. Отстранившись от свойственного человеку метода восприятия мира посредством органов чувств, посмотрим на процессы вокруг нас с точки зрения чистой информации. Так, мы увидим, что все процессы по своей сути описывают систему передачи математических параметров между любыми, исследуемыми человеком, объектами. Информатика, по своей сути, и есть система отношений физических (а на наиболее глубоких уровнях и математических) параметров. Человек в данном случае стремится измерить данные отношения, понять их природу, и использовать в личных целях. Таким образом информатика не отождествляема от разумной деятельности человека. По ходу совершенствования технологических способностей человечества, методы работы с информацией неустанно совершенствовались. Благодаря появлению и развитию интегрированных сред разработки (далее ИСР), работа с информацией стала как некогда интуитивна и проста. Поэтому целью данной работы я ставлю изобразить решение классических проблем информатики с помощью среды программирования, максимально понятно донести до читателя основные положения и проблемы такой научной области, как теоретической информатики и дать базовые знания для работы на языке программирования python.

Основная часть:

Глава 1: Понятие энтропии, измерение количества информации.

Раздел 1: Информация, энтропия в теории информации.

В изучении окружающего нас мира и присущих ему процессов, мы не рано или поздно встречаемся с понятием информации.

Можно выделить несколько определений понятия информации (Madden, 2000):

1. Информация как представление знаний. Информация – есть хранимое на определенном носителе знание.
2. Информация как данные в окружающем нас мире. Информация – интерпретируемые человеком природные феномены.
3. Информация как часть коммуникационного процесса. Информация – как вербальные и невербальные сообщения передаваемые между людьми .
4. Информация как ресурс и экономическое благо. Информация – сфокусированное сообщение, имеющее получателя и отправителя, содержащее в себе определенную ценность для обеих сторон, и способное к произведению собой прибавочной стоимости в определенных контекстах.

Стремление понять и обуздать информацию привело к развитию теории информации, центральным элементом которой служит математическая теория коммуникации Шеннона (SHANNON, 1948):

В ней, общая коммуникационная система представляется в виде:

В 1928 году, была издана научная статья американского ученого Ральфа Винстона Лайона Хартли (HARTLEY, 1928), в ней он предлагает использовать для измерения количества информации в сообщении, логарифмическую меру вида:

I = K log2 N

Где:

I - количество информации в сообщени в битах

N – количество символов в используемом алфавите

K – длина сообщения

В 1948 году Шеннон, в своей работе “математическая теория коммуникации” (SHANNON, 1948), произвел прорыв в такой науке, как теория информации, введя определение информационной энтропии.

При отсутствии информационных потерь, энтропия рассчитывается по формуле Хартли.

Однако, при наличии неопределенности, средняя энтропия сообщения вычисляется по формуле:

H(x) = - Σ pi log2 pi

Где частная энтропия сообщения:

Hi = - log2 pi

Раздел 2: нахождение количества информации в сообщении с помощью языка программирования python.

Все выше предоставленные способы нахождения имеют один главный общий элемент, это двоичный логарифм. Проблема, стоящая перед нами, заключается в том, что сам по себе python не имеет функции log, так как она перенесена в модуль math.

Так как цель данной работы в первую очередь показать, как используя только базовый функционал языка программирования python, возможно решить классические проблемы теоретической информатики, то все функции будут написаны самостоятельно, и позже использованы для решения более высокоуровневых проблем.

Есть много способов написать функцию нахождения логарифма на языке программирования, перечислим их:

1. С помощью алгоритма двоичного поиска
2. С помощью рекурсии
3. С помощью нахождения натурального логарифма и последующего приведения его к другому основанию:
   1. Используя ряд Тэйлора
   2. Приближенно, ln(x) = lim n→∞ n(x1/n - 1)

Все методы, приведенные выше хороши, но в нашем случае нам не нужна абсолютная точность вычислений, для нас важны две характеристики: простота в использовании и скорость.

Использовать мы будем следующие формулы:

1. Logb(x) = logk(x) / logk(b)= ln(x) / ln(b)
2. ln(x) = lim n→∞ n\*(x1/n - 1)

На языке питон мы можем представить два данных уравнения с помощью следующих функций:

def ln(x):

    n=100000

    return n\*(x\*\*(1/n)-1)

def log(base, x):

    return ln(x)/ln(base)

- так как мы не можем использовать бесконечность в вычислениях, то возьмем приближенно такое число n, которое позволит нам без ошибки получить логарифм с основание два, возвращающий 64, для этого функция log(2, 1.8446744e+19) должна вернуть нам 64 с точностью до сотых.

Используя n=100000, мы получаем следующий результат:

print(log(2, 1.8446744e+19))

-----------------------------------

64.0139758927805

n=100000 удовлетворяет нашей потребности

Получив функцию логарифма, мы можем составить все функции энтропии:

Формула Хартли:

def mesgInf(mesgLength, alphabetLength):

return mesgLength\*log(2, alphabetLength)

mesgLenght – длина сообщения

alphabetLenght - количество символов в используемом алфавите

Формула Шеннона:

Частная энтропия:

def entropy(signal,noise):

return -log(2, signal/noise)

signal – полезная информация

noise – шум

Средняя энтропия сообщения:

def totalEntropy(states):

return sum(map(lambda x: x[0]/x[1]\*entropy(x[0],x[1]), states))

states – двухмерная матрица, вида:

|  |  |  |
| --- | --- | --- |
| Номер строки | x[0] - полезная информация | x[1] – шум |
| 1 | … | … |
| … | … | … |
| n | … | … |

Так как данные функции чаще всего будут выдавать дробное число бит, а для практического использования нам необходимо целое число бит, то необходимо округлить результаты вверх. Проблема заключается в том, что python не имеет встроенной функции округления вверх. Однако, мы можем использовать оператор //, который всегда возвращает результат деления без остатка, а то есть исполняет округление вниз, и использовать его на числе Y, противоположное по знаку искомому числу X.

def roundUp(num):

    return -(-num//1)

Благодаря тому, что мы округляем вниз отрицательное число, то просто поменяв знак на противоположный, мы получим результат округления противоположного числа вверх.

Примеры:

1.

print(mesgInf(5, 33))

--------------------------

25.22232413228775

print(roundUp(25.22232413228775))

-----------------------------------------

26.0

-бит нужно для кодирования сообщения длиною в пять символов, с помощью маленьких букв русского алфавита. Так как ответ десятичный, то для практического использования результата, его нужно округлить в большую сторону. Ответ: 26 бит.

2.

print(entropy(10,100))

---------------------------

3.3218783373474965

print(roundUp(3.3218783373474965))

------------------------------------------

4.0

- бит нужно для выражения меры неопределенности сообщения при отношении полезной информации к шуму 10/100. Для применения на практике ответ стоит округлить в большую сторону. Ответ: 4 бит.

3.

print(totalEntropy([[1,8],[7,8]]))

------------------------------------------

0.5435585479019456

print(roundUp(0.5435585479019456))

------------------------------------------

1.0

-бит нужно для выражения меры неопределенности двух вариантов событий, 1/8 и 7/8. Вводные данные можно представить в виде матрицы:

|  |  |  |
| --- | --- | --- |
| Номер варианта | Полезная информация | Шум |
| 1 | 1 | 8 |
| 2 | 7 | 8 |

Для применения на практике округлим результат в большую сторону, и получим 1 бит. Ответ: 1 бит.

Глава 2: Системы счисления.

Раздел 1: Теория переводов систем счисления. Машинный код.

Имея способ найти количество информации, перед нами встает вопрос практического применения полученного двоичного числа. Первое, что приходит на ум, это перевод двоичного числа в десятичное. Данный перевод можно произвести с помощью метода Горнера, изложенного им самим в статье “A New Method of Solving Numerical Equations of All Orders, by Continuous Approximation” в журнале “Philosophical Transactions of the Royal Society of London” в 1819 году (Horner, 1819). Для преобразования надо суммировать числа слева на право, умножая раннее полученный результат на основание системы счисления (в случае двоичной системы счисления, данное основание будет равно 2).

Однако, способ приведенный выше достаточно неудобно применить в программировании, так что мы используем схему, при которой каждый разряд числа в n-ой системе счисления умножается на номер разряда справа налево начиная с нуля, и результат суммируется.

Если число дробное, то его дробная часть обрабатывается согласно тому же алгоритму, только каждый элемент дробной части умножается на основание в степени хода от запятой до конечного элемента, начиная с нуля и умножаясь на минус 1.

Имея способ повышения системы счисления, встает проблема понижения. Для решения данной задачи используем следующий алгоритм: будем делить десятичное число на основание новой системы счисления, каждый раз убирая остаток от деления и записывая его отдельно, пока не получим ноль. Тогда результатом перевода будет запись всех остатков от деления, в обратном порядке.

Для дробного числа, дробная часть в меньшей системе счисления находится путем умножения дробной части на новое основание, пока целая часть не будет больше нуля. Целые части результатов умножения записываются после запятой у нового числа, а дробная часть продолжает умножаться, пока не будет достигнут желаемый уровень точности перевода.

Работая с компьютерными системами, большая часть данных представляется в двоичном коде. Отсюда появляется нужда в проведении арифметических операций над двоичными числами. Существование отрицательных чисел и такой операции, как вычитание усложняет задачу бинарных операций, так как одного сумматора становится недостаточно. Однако, в 1945 году Джон Фон Нейман в первом проекте отчета о EDVAC (Neumann, 1945), принес в мир идею о “дополнительном коде”, используя который можно представлять как положительные, так и отрицательные числа с фиксированной точкой в двоичной системе счисления.

Кроме прочего, стоит иметь способ перевода двоичных чисел в 16 систему счисления, так как она широко используется в компьютерных системах благодаря простоте конвертации из 16 в 2 и обратно. Во многих случаях, шестнадцатеричная система наиболее рациональна из всех остальных. Для перевода используем метод тетрад.

Раздел 2: Перевод систем счисления, суммирование двоичных чисел, перевод в дополнительный код на языке python.

Перевод из степени счисления меньше 10, в десятичную мы можем выполнить с помощью следующей функции:

def incrCC(num, cc):

    output=0

    num=str(num)

    if '.' in num:

        whole,fract=num.split('.')

        for i in range(len(fract)):

            output+=int(fract[i])\*cc\*\*(-(i+1))

            num=whole

    leng=len(num)

    for i in range(leng):

        output+=int(num[i])\*cc\*\*(leng-i-1)

    return output

num - число в системе счисления n

cc – основание системы счисления n, сс = n

Алгоритм будет поразрядно умножать каждый разряд на основание системы счисления, возведенной в степень равную номеру разряда справа налево для целой части, и слева на право для дробной.

Перевод из десятичной системы счисления в меньшую мы изобразим так:

def decrCC(num,cc, precision):

    output=''

    if '.' in str(num):

        whole,fract=str(num).split('.')

        fract='0.'+fract

        while precision>0:

            fract=float(fract)\*2

            fract=str(fract).split('.')

            output=output+fract[0]

            fract='0.'+fract[1]

            precision=precision-1

        output='.'+output

        num=int(whole)

    while num!=0:

        output=str(num%cc)+output

        num=num//cc

    return output

num – число в 10 системе счисления

cc – основание новой системы счисления

precision – точность, до которой нужно вычислить дробную часть нового числа

Для нахождения целой части алгоритм дописывает остаток от деления на основание системы счисления слева, делит число на основание без остатка, и повторяет алгоритм, пока число не будет равно 0. Для нахождения дробной части в новой системе счисления, алгоритм умножает ее на основание системы счисления, пока результат не будет больше или равен 1. С каждым умножением целая часть приписывается справа, пока точность не достигнет желаемой.

Примитивное суммирование двух двоичных чисел мы можем максимально кратко и эффективно записать следующим образом:

def binSum(num1,num2):

    output = str(num1+num2).replace('12','100').replace('2','10')

    return output

num1 – первое число в двоичной системе

num2 – второе число в двоичной системе

Особенность алгоритма в том, что он воспринимает двоичные числа как десятичные, что позволяет очень быстро сложить два числа, без всех условий, присущих либо суммации напрямую в двоичной системе, либо в переводе чисел в десятичную при суммации, а потом обратно в двоичную. Числа складываются как есть, ‘12’ в результате заменяется на ‘100’, а ‘2’ на ’10’.

В дополнительном коде производится инверсия для отрицательного числа, так напишем функцию, инвертирующую знаки двоичного числа:

def polar(num):

    newNum=''

    switch = {'0':'1','1':'0'}

    for el in num:

        newNum=newNum+switch[el]

    return newNum

num – изначальное число в двоичной системе счисления

Алгоритм поразрядно меняет знаки на противоположные и приписывает их к ответу справа.

Для получения дополнительного кода десятичного числа воспользуемся раннее написанными функциями polar(), binSum() и decrCC():

def directCode(num):

    code=decrCC(abs(num),2)

    leng=len(code)

    if abs(num)<128:

        byteNum=8

    else:

        byteNum=16

    dif=byteNum-leng%byteNum-1

    code='0'\*dif+code

    if num<0:

        return binSum(int('1'+polar(code)),1)

    else:

        return '0'+code

num – число в десятичной системе счисления

Для начала мы переводим число в двоичную систему счисления, потом мы находим длину полученного двоичного числа, если число можно представить в с помощью одного байта, то искомая длина кода будет равна 8, иначе 16(это позволяет представить числа, которые по модулю меньше 32768, для представления больших чисел, алгоритм можно расширить для 3 байт, 4 байт и т.д.) Если число в двоичном формате не полностью занимает все биты кроме знакового, то приписываем слева нули, пока длина числа не будет равна искомой длине - 1. Если число отрицательное, то знаки инвертируются, прибавляется 1, а знаковый разряд равен единице. Иначе, знаковый разряд равен нулю, а двоичная форма числа никак не меняется.

Для получения 16-ого числа из 2-ого напишем следующую функцию:

def hexdecCC(num):

    output=''

    num=str(num)

    dic={'0000':'0','0001':'1','0010':'2','0011':'3','0100':'4','0101':'5','0110':'6','0111':'7','1000':'8','1001':'9','1010':'A','1011':'B','1100':'C','1101':'D','1110':'E','1111':'F'}

    dif = len(num)%4

    if dif:

        num='0'\*(4-dif)+num

    while num!='':

        output=output+dic[num[0:4]]

        num=num[4:]

    return output

num – число в 2-ой системе счисления

Алгоритм для начала проверит, делится ли длина двоичного числа ровно на 4, если нет, то он допишет недостающие нули. Дальше, каждые 4 символа слева на право будут сопоставляться со значениями в словаре и согласно их представлению в 16-ой системе счисление приписываться справа в результирующем числе.

Для обратной операции, используем похожую функцию:

def binHexdec(num):

    output=''

    num=str(num)

    dic={'0':'0000','1':'0001','2':'0010','3':'0011','4':'0100','5':'0101','6':'0110','7':'0111', '8':'1000', '9':'1001', 'A':'1010', 'B':'1011', 'C':'1100', 'D':'1101', 'E':'1110', 'F':'1111'}

    for el in num:

        output=output+dic[el]

    return output

num – число в 16-ой системе счисления

Перевод из двоичной в восьмеричную и обратно аналогичен:

def hexCC(num):

    output=''

    num=str(num)

    dic={'000':'0','001':'1','010':'2','011':'3','100':'4','101':'5','110':'6','111':'7'}

    dif = len(num)%3

    if dif:

        num='0'\*(3-dif)+num

    while num!='':

        output=output+dic[num[0:3]]

        num=num[3:]

    return output

num – число в 2-ой системе счисления

def binHex(num):

    output=''

    num=str(num)

    dic={'0':'000','1':'001','2':'010','3':'011','4':'100','5':'101','6':'110','7':'111'}

    for el in num:

        output=output+dic[el]

    return output

num – число в 8-ой системе счисления

Глава 3: совершенный дизъюнктивный и конъюнктивный коды на языке python.

Алгебра логики является фундаментом работы любого цифрового автомата. Конъюнкция и дизъюнкция служат главными инструментами в построении логических выражений, поэтому составление элементарных конъюнкций и дизъюнкций, а с их помощью и совершенных конъюнктивных и дизъюнктивных кодов (СКНФ, СДНФ), является одной из базовых проблем информатики.

Эти коды могут быть достаточно просто составлены на языке python.

def conjunction(table):

    output=''

    for line in table:

        out=''

        res = line.pop()

        if res==0:

            for i in range(len(line)):

                if line[i]==1:

                    out+='not(x'+str(i+1)+')+'

                else:

                    out+= 'x'+str(i+1)+'+'

            output=f'{output}\*({out[:-1]})'

        print(line)

    return output[1:]

table – таблица истинности

Алгоритм для получение СКНФ из таблицы истинности предоставленной в виде двухмерной матрицы.

def disjunction(table):

    output=''

    for line in table:

        out=''

        res = line.pop()

        if res==1:

            for i in range(len(line)):

                if line[i]==0:

                    out+= 'not(x'+str(i+1)+')\*'

                else:

                    out+= 'x'+str(i+1)+'\*'

            output=f'{output}+{out[:-1]}'

        print(line)

    return output[1:]

table – таблица истинности

Алгоритм для получения СДНФ.

Заключение:

Развитие сред программирования безусловно упрощает решение классических задач теоретической информатики. Своей работой я стремлюсь продемонстрировать то, как программирование с помощью высокоуровневого языка может помочь в понимании теоретических основ программирования, а также их наглядной демонстрации. Данные мною решения проблем, я истинно верю, могут быть полезны в преподавании такой дисциплины, как “Теоретические основы программирования”, а также они могут быть использованы как соединяющее звено между теорией информатики и ее практическим применением.

# Список литературы

[HARTLEY, R. V. (1928). Transmission of Information. *Nokia Bell Labs*, 535-563.](https://monoskop.org/images/a/a6/Hartley_Ralph_VL_1928_Transmission_of_Information.pdf)

[Horner, W. G. (1819). A New Method of Solving Numerical Equations of All Orders, by Continuous Approximation. *Philosophical Transactions of the Royal Society of London*, 308-335.](https://course.ece.cmu.edu/~ece447/s15/lib/exe/fetch.php?media=horner-1819.pdf)

[Madden, A. D. (2000). A definition of information. *Aslib Proceedings*, 343-349.](https://eeclass.ntsu.edu.tw/sysdata/doc/3/39659056a8757b94/pdf.pdf)

[Neumann, J. v. (1945). First Draft of a Report on EDVAC.](http://abelgo.cn/cs101/papers/Neumann.pdf)

[SHANNON, C. E. (1948). A Mathematical Theory of Communication.](https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf) *[The Bell System Technical Journal](https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf)*[, 379-423, 623-656.](https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf)