|  |  |
| --- | --- |
|  | **Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет имени Н.Э. Баумана**  **(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**ОТЧЕТ ПО УЧЕБНО-ВЫЧИСЛИТЕЛЬНОМУ**

**ПРАКТИКУМУ**

Студент Ляпина Наталья Викторовна

*фамилия, имя, отчество*

Группа ИУ7-12Б

Тип практики Учебная

Название предприятия\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент  Ляпина Н.В.  *подпись, дата фамилия, и.о.*

Руководитель практики  Борисов С. В.   *подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2020 г.*

**Оглавление**

1. Условие задачи…………………………………...…………………………3

1. Схема программы…………………………………………………………...4

1. Описание программы………………………….…………………………...11

1. Текст программы…………………………….……………………………..15

1. Заключение…………………………….………………………………........18
2. Список литературы…………...……………………….…………………....18

**Текст задачи**

**Программа проверяющая анаграммы**

Часто забавно видеть как рекомбинация букв названия дает забавные анаграммы. Например, буквы из словосочетания «WILLIAM SHAKESPEARE» перестраиваются в «SPEAK REALISM AWHILE».

Напишите программу, которая будет считать словарь и список фраз и определять какие слова из словаря будут анаграммой данных фраз. Ваша программа должна найти все наборы слов в словаре, которые могут быть сформированы из букв каждой фразы. Не включайте набор, состоящий из первоначальных слов. Если никакой анаграммы не найдено, то не печатайте ни чего, даже незаполненную строку.

**Ввод**

Ввод будет состоять из двух частей. Первая часть – словарь, вторая часть – набор фраз, для которых потребуется найти анаграммы. Каждая часть файла будет оканчиваться строкой, состоящей из единственного символа #. Словарь задается в алфавитном порядке и будет содержать до 2000 слов, по одному на каждой строке. Весь файл будет в верхней регистре, и никакое слово в словаре или фраза не должны содержать более 20 символов. Вы можете использовать только Английский язык.

**Вывод**

Вывод будет состоять из нескольких строк. Каждая строка будет состоять из первоначальной фразы, пробела, знака(=), другого пробела, и списка слов, которые составляют анаграмму первоначальной фразы, разделенных пробелом. Эти слова должны быть в алфавитной последовательности.

**Пример ввода**

ABC

AND

DEF

DXZ

K

KX

LJSRT

LT

PT

PTYYWQ

Y

YWJSRQ

ZD

ZZXY

#

ZZXY ABC DEF

SXZYTWQP KLJ YRTD

ZZXY YWJSRQ PTYYWQ ZZXY

#

**Пример вывода**

SXZYTWQP KLJ YRTD = DXZ K LJSRT PTYYWQ

SXZYTWQP KLJ YRTD = DXZ K LT PT Y YWJSRQ

SXZYTWQP KLJ YRTD = KX LJSRT PTYYWQ ZD

SXZYTWQP KLJ YRTD = KX LT PT Y YWJSRQ ZD

**Схема программы**

Основная программа

Начало

Цикл ПОКА:

Вводимые данные для набора не будут равны: “ # ”

А

В

Цикл for:

Перебор всех фраз из массива с фразами

Цикл ПОКА:

Вводимые данные для набора не будут равны: “#”

0

Обращение к функции full\_Anagrams (массив фраз, массив слов)

Обращение к функции, которая проверяет фразу по полное соответствие исходному словарю

----

Нет

Да

Фраза состоит из слов словаря

Добавление данных в массивы

Ввод данных наборов слов и фраз

В

Answer=combinated\_Anagrams (delite\_space(фраза), anagrams\_Dict (фраза,исходный словарь))

А

Вывод массива Answer и фразы

Цикл for:

Перебор всех фраз из массива с фразами

Конец

Функция full\_Anagrams

full\_Anagmas (main\_Phrase, dictionary)

----

Начало

Цикл for:

Перебор всех символов фразы

Показатель присутствия слова фразы в словаре

----

counter=0

Если символ не = пробелу

Нет

Да

Цикл for:

Перебор всех слов словаря

word+=символ

word = слово словоря

Нет

Да

counter = 1

Цикл for:

Перебор всех слов словаря

C

D

Нет

Да

Counter=0

C

D

break

word=пустая строка

Цикл for:

Перебор всех символов фразы

counter>0

Конец

Функция combinated\_Anagrams



combinated\_Anagrams (phrase, dict)

Начало

combinatinons=[]

Обращение к функции Binary, которая возвращает двоичное представление числа ((2 в степени длина dict) -1) в виде строки

n=Binary(pow(2,len(dict))-1)



Цикл for:

Для i от 1 до 2 в степени len(dict)

b=Binary(i)

– двоичное число в виде строки

– составленная фраза в виде строки

– та же фраза только в виде массива с отдельными словами

– длина составленной фразы

b='0'\*(len(n)-len(b))+b

combinated\_phrase\_str=''

combinated\_phrase\_array=[]

len\_Combination=0



E

Цикл for:

Перебираем элементы строки b

E

Элемент строки b =’1’

Нет

Да

combinated\_phrase\_str += слово из словаря под номером элем

combinated\_phrase\_array.append (слово из словаря под номером элем)

len\_Combination+= длина слова из словаря под номером элем

Цикл for:

Перебираем символы строки b

Длина

комбинации = длине фразы

Нет

Да

Комбинация и фраза сост. из одинаковых букв

Нет

Да

Добавляем в массив с новыми фразами массив с комбинацией слов

F

Цикл for:

Для i от 1 до 2 в степени len(dict)

F

Выводим массив с новыми фразами

Конец

Функция anagrams\_Dict



Начало

anagrams\_Dict(phrase, dict\_Array)



k – количество совпавших букв во фразе и слове

anagrams= [ ]

k=0

Обращаемся к ф-ции letters\_Counter, которая возвращает словарь с буквами слова и их количеством



phrase\_letters= letters\_Counter (phrase)

Цикл for:

Перебор всех слов массива dict\_Array

new\_word= letters\_Counter (слово массива)

Цикл for:

Перебор всех букв слова new\_word

Кол-во букв под номером i во фразе минус кол-во букв под номером i в new\_word >=0

Нет

Да

k+=1

H

G

G

H

Цикл for:

Перебор всех букв слова new\_word

Все буквы new\_word входят во фразу

Нет

Да

Добавляем слово в массив anagrams

k=0

Цикл for:

Перебор всех слов массива dict\_Array

len(anagrams) !=0

Нет

Да

anagrams

Конец

Функция letters\_Counter

letters\_Counter(word)

----

Начало

word\_length=len(word)

letters={}

word=''.join(sorted(word))

сортируем все буквы в слове (фразе) в алфавитном порядке и создаем словарь

----

Цикл for:

i от 0 до word\_length

Символ под номером i = пробелу

Нет

Да

Символ уже есть в словаре

Нет

Да

Добавляем в словарь letters новый ключ «символ» и его значение 1

Прибавляем к значению ключа «символ» единицу в словаре letters

Цикл for:

i от 0 до word\_length

Словарь letters

Конец

**Описание программы**

**Основная программа**

Задаются два пустых массива: **dictionary\_Array**, **phrases\_Array**. В эти массивы последовательно записываются сначала слова для словаря в массив **dictionary\_Array**, затем фразы в массив **phrases\_Array**. Вводимые пользователем данные записываются в переменные **dictionary\_Word** и **phrase\_Word** соответственно.

После записи всех входных данных программа входит в цикл for, который проходится по всему массиву с фразами **phrases\_Array**, обращаясь по очереди к каждому его элементу. Далее, если фраза является полной анаграммой (т.е все слова этой фразы входят в массив **dictionary\_Array**); то ничего не выводится и цикл обращается к следующей фразе.

В ином случае программа обращается к функции **combinated\_Anagrams**, которая составляет все возможные комбинации подходящих слов и записывает их в массив. Далее составляющие этого массивы выводятся в порядке, заданном условием задачи.

**Функция letters\_Counter**

**Задача**: составить из введенной строки словарь, который содержит информацию о количестве элементов этой строки

**Входные данные**: переменная word тип str

**Алгоритм**:

В самом начале функции создаются две переменные: **word\_length** – длина вводимой строки, **letters** – пустой словарь, который будет содержать символы строки и их количество. Переменная **word** перезаписывается в строку, которая содержит все символы введенной строки **word** , отсортированные в алфавитном порядке методом *word=''.join(sorted(word)).*

Затем программа входит в цикл for, который в качестве аргумента использует длину введенной строки – переменную **word\_length**. В каждом цикле программа обращается к переменной **word** по индексу **i** и проверяет является ли этот элемент пробелом, чтобы избежать записи пробелов в словарь.

Далее элемент либо добавляется в словарь в качестве ключа, получая значение 1, если его еще нет в словаре; либо если этот элемент уже присутствует в словаре, то к его значению прибавляется единица.

Выходные данные: {‘a’: 2, ‘b’: 1, ‘h’:2, ‘z’:3}.

Такое хранение данных крайне удобно для проверки вхождения букв слова во фразу, т. к. при сравнении слова и фразы не обязательно проходиться по всей фразе, чтобы узнать являются ли они анаграммами, достаточно просто сравнить словари.

**Функция anagrams\_Dict**

**Задача:** из исходного массива слов составить новый массив, который будет включать в себя только те слова, которые являются анаграммами к исходной фразе

**Входные данные:** phrase, dict\_Array

**Алгоритм:**

На вход программе дается исходная фраза, записанная в переменную **phrase**, и исходный словарь, записанный в переменную **dict\_Array.**

В начале программы создаются новые переменные: **anagrams** – массив нового словаря, **k** – счетчик количества букв слова, которые входят во фразу, **phrase\_letters** – словарь со всеми буквами фразы (получается путем обращения к функции **letters\_Counter**).

Затем в цикле for каждое слово исходного словаря проверяется на анаграмму с исходной фразой, и если фраза содержит все буквы слова, то это слово записывается в новый массив anagrams.

В конце программа проверяет длину массива anagrams, и если она не равна 0, то возвращает его.

**Функция dict\_compare**

**Задача**: сравнить два словаря на полное соотвествие

**Входные данные** : dict1 dict2

**Алгоритм**:

Создается новая переменная **comp**, которая содержит количество совпавших пар ключ – значение.

Сначала сравниваются массивы ключей двух словарей, если они не равны, то возвращается значение *False*. В ином случае сравниваются значения этих ключей в обоих словарях, и если они равны, то к переменной comp прибавляется 1.

Если количество совпавших пар **comp** равно количеству ключей первого словаря, то возвращается значение *True*, а если нет, то *False*.

**Функция Binary**

**Задача**: перевести вводимое число в двоичную систему и представить его в виде строки

**Функция combinated\_Anagrams**

**Задача:** составить все возможные комбинации из исходных слов, содержащихся в массиве, которые являются анаграммами к исходной фразе.

**Входные данные:** phrase, dict

**Алгоритм:**

Создается пустой массив **combinations**, который будет содержать проверенные на соответствие комбинации. Затем в переменную **n** (путем обращения к функции **Binary**) записывается двоичное представление длины массива **dict**, который содержит все слова, являющиеся анаграммами к фразе **phrase**.

Затем программа входит в цикл for, в котором переменная **i** принимает значения от 1 до 2 в степени длины массива **dict**. Новая переменная **b** принимает значение i (путем обращения к функции Binary), переведенное в двоичную систему. При этом незначащие нули перед числом сохраняются.

Создаются еще три новые переменные: **combinated\_phrase\_str** – строка, полученная путем комбинаций элементов массива **dict**, **combinated\_phrase\_array** – массив, который содержит эту же строку, только разбитую на отдельные слова, **len\_Combination** – переменная, содержащая длину полученной строки.

После программа входит во второй цикл for, который проходится по строке b, содержащей двоичное число. Если элемент строки b равен ‘1’, то слово под этим номером добавляется к строке **combinated\_phrase\_str** , и в массив **combinated\_phrase\_array**, а его длина добавляется в переменную **len\_Combination.**

Затем длина комбинации проверяется на соответствие длине исходной фразы **phrase**, и если они равны, то сравниваются словари с буквами каждой из строк **combinated\_phrase\_str** и **phrase**. Если и они равны, то комбинация **combinated\_phrase\_array** добавляется в массив комбинаций **combinations**.

Чем удобная такая система поиска комбинаций?

Сам алгоритм, по сути, представляет таблицу истинности, где 0 – отсутствие слова в комбинации, а 1 – присутствие. Таким образом, при просмотре комбинаций мы не зависим от количества слов в комбинации, и нам не приходится рассматривать повторные комбинации. Также нам не приходится расставлять комбинации в алфавитном порядке для вывода, потому что в исходном массиве dict они находятся в алфавитном порядке, а составление комбинаций начинается с конца массива. Следовательно, при выводе двумерного массива в конце основной программы нам достаточно перевернуть массив и вывести комбинации.

**Функция full\_Anagrams**

**Задача:** проверить состоит ли фраза полностью из слов исходного словаря.

**Входные данные**: main\_Phrase, dictionary

**Алгоритм:**

Создается переменная **word**, в которую записывается каждое отдельное слово фразы **main\_Phrase**. Затем переменная **flag**, которая будет хранить информацию о том, содержится слово фразы в массиве dictionary или нет. После этого в конец строковой переменной **main\_Phrase** добавляется пробел, чтобы последнее слово фразы тоже вошло в цикл для сравнения.

Программа входит в первый цикл for, который проходится по элементам фразы **main\_Phrase**. Пока элемент не пробел, он добавляется к переменной **word**, тем самым разбивая фразу на отдельные слова. Как только нашелся пробел, программа входит во второй цикл for, который проходится по всему массиву **dictionary** и сравнивает слово **word** со словом из этого массива. Если такое слово есть, то **flag** равен 1. Как только второй цикл заканчивается, то **flag** проверяется на соответствие, если он равен 0, то программа выходит из первого цикла. Если нет, то **word** приравнивается к пустой строке.

Функция возвращает значение **flag**>0, которое равно *False*, если фраза не является полной анаграммой; и значение *True*, если является.

# Текст программы

# Практика 1 семестр

# Ляпина Наталья Викторовна ИУ7-12Б

# ПЕРЕМЕННЫЕ

# массивы

dictionary\_Array=[]

phrases\_Array=[]

# строки

dictionary\_Word=''

phrase\_Word=''

# Определение количества различных букв в слове или фразе

def letters\_Counter(word):

word\_length=len(word)

letters={}

word=''.join(sorted(word))

for i in range(word\_length):

if word[i]!=' ':

if letters.get(word[i])==None:

letters[word[i]]=1

else:

letters[word[i]]+=1

return letters

# создание словаря с анаграммами для фразы

def anagrams\_Dict(phrase,dict\_Array):

anagrams=[]

k=0

phrase\_letters=letters\_Counter(phrase)

for word in dict\_Array:

new\_word=letters\_Counter(word)

for i in (new\_word.keys()):

if phrase\_letters.get(i)!=None and phrase\_letters[i]-new\_word[i]>=0:

k+=1

if k==(len(new\_word.keys())):

anagrams.append(word)

k=0

if len(anagrams)!=0:

return anagrams

# удаление пробела во фразе

def delite\_space(phrase):

K=''

for letter in range(len(phrase)):

if phrase[letter]!=' ':

K+=phrase[letter]

return K

# проверка двух словарей на идентичность

def dict\_compare(dict1,dict2):

comp=0

if dict1.keys()!= dict2.keys():

return False

else:

for i in dict1:

if dict1[i]==dict2[i]:

comp+=1

if comp==len(dict1.keys()):

return True

else:

return False

# перевод в двоичную систему счисления

def Binary(k):

b=''

while k>0:

b=str(k%2)+b

k=k//2

return b

# перебор всех возможных комбинаций для словаря с анаграммами

def combinated\_Anagrams(phrase,dict):

combinatinons=[]

n=Binary(pow(2,len(dict))-1)

for i in range(1,pow(2,len(dict))):

b=Binary(i)

b='0'\*(len(n)-len(b))+b

combinated\_phrase\_str=''

combinated\_phrase\_array=[]

len\_Combination=0

for j in range(len(b)):

if b[j]=='1':

combinated\_phrase\_str+=dict[j]

combinated\_phrase\_array.append(dict[j])

len\_Combination+=len(dict[j])

if len\_Combination==len(phrase):

if dict\_compare(letters\_Counter(phrase),letters\_Counter(combinated\_phrase\_str)

)==True:

combinatinons.append(combinated\_phrase\_array)

return combinatinons

# проверка фразы на полную анаграмму

def full\_Anagrams(main\_Phrase,dictionary):

word=''

flag=0

main\_Phrase+=' '

for i in range(len(main\_Phrase)):

flag=0

if main\_Phrase[i]!=' ':

word+=main\_Phrase[i]

else:

for j in range(len(dictionary)):

if word==dictionary[j]:

flag=1

if flag==0:

break

word=''

return(flag>0)

# Ввод словаря

while dictionary\_Word!='#':

dictionary\_Word=input()

if dictionary\_Word!='#':

dictionary\_Array.append(dictionary\_Word)

# Ввод фраз

while phrase\_Word!='#':

phrase\_Word=input()

if phrase\_Word!='#':

phrases\_Array.append(phrase\_Word)

# перебор всех анаграмм для фразы

for i in range(len(phrases\_Array)):

Answer=[]

if full\_Anagrams(phrases\_Array[i],dictionary\_Array)==False:

Answer=combinated\_Anagrams(delite\_space(phrases\_Array[i]),anagrams\_Dict(

phrases\_Array[i],dictionary\_Array))

# вывод

Answer.reverse()

for j in range(len(Answer)):

print(phrases\_Array[i],end =' = ')

for k in Answer[j]:

print(k,end=' ')

print()

# Заключение

В ходе практикума я научилась искать оптимальные пути решения задач, используя свои знания. Также, для решения данной задачи, я изучила комбинаторику и углубилась в структуру языка python

# Список литературы

1. Лекции по курсу Борисов С. В.
2. <https://habr.com/ru/post/494650/>