Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени федеральное государственное

бюджетное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и

информационных технологий

Лабораторная работа №3

на тему: «Методы поиска подстроки в строке»

Выполнила:

Студентка группы БФИ1902

Струкова А.В.

Вариант 18

Проверил:

Москва, 2021 г.

Оглавление

[1. Цель работы 2](#_Toc72409356)

[2. Задание на лабораторную работу 2](#_Toc72409357)

[3. Листинг программы 2](#_Toc72409358)

[4. Результат работы программы 7](#_Toc72409359)

[Список использованных источников 8](#_Toc72409360)

# **1. Цель работы**

Цель работы: рассмотреть и изучить основные методы поиска подстроки в строке.

# **2. Задание на лабораторную работу**

**Задание 1**

Реализовать методы поиска подстроки в строке. Добавить возможность ввода строки и подстроки с клавиатуры. Предусмотреть возможность существования пробела. Реализовать возможность выбора опции чувствительности или нечувствительности к регистру. Оценить время работы каждого алгоритма поиска и сравнить его со временем работы стандартной функции поиска, используемой в выбранном языке программирования. Алгоритмы: 1.Кнута-Морриса-Пратта 2.Упрощенный Бойера-Мура

**Задание 2 «Пятнашки»**

# **3. Листинг программы**

*# -\*- coding: utf8 -\*-*

*#Кнута-морриса-прата*

lil = "лилила"

p = [0]\***len**(lil)

j = 0

i = 1

while i < **len**(lil):

    if lil[j] == lil[i]:

        p[i] = j+1

        i += 1

        j += 1

    else:

        if j == 0:

            p[i] = 0;

            i += 1

        else:

            j = p[j-1]

**print**(p)

st = "лилилось лилилась"

lent = **len**(lil)

lenst = **len**(st)

i = 0

j = 0

while i < lenst:

    if st[i] == lil[j]:

        i += 1

        j += 1

        if j == lent:

**print**("образ найден")

            break

    else:

        if j > 0:

            j = p[j-1]

        else:

            i += 1

if i == lenst:

**print**("образ не найден")

*#Урощенный Бойера-Мура*

data = "данные"

*# Этап 1: формирование таблицы смещений*

S = **set**() *# уникальные символы в образе*

M = **len**(data) *# число символов в образе*

d = {} *# словарь смещений*

for i in **range**(M-2, -1, -1): *# итерации с предпоследнего символа*

    if data[i] not in S: *# если символ еще не добавлен в таблицу*

        d[data[i]] = M-i-1

        S.**add**(data[i])

if data[M-1] not in S: *# отдельно формируем последний символ*

    d[data[M-1]] = M

d['\*'] = M              *# смещения для прочих символов*

**print**(d)

*# Этап 2: поиск образа в строке*

weather = "метеоданные"

N = **len**(weather)

if N >= M:

    i = M-1       *# счетчик проверяемого символа в строке*

    while(i < N):

        k = 0

        j = 0

        for j in **range**(M-1, -1, -1):

            if weather[i-k] != data[j]:

                if j == M-1:

                    off = d[weather[i]] if d.**get**(weather[i], False) else d['\*']  *# смещение, если не равен последний символ образа*

                else:

                    off = d[data[j]]   *# смещение, если не равен не последний символ образа*

                i += off    *# смещение счетчика строки*

                break

            k += 1          *# смещение для сравниваемого символа в строке*

        if j == 0:          *# если дошли до начала образа, значит, все его символы совпали*

**print**(f"образ найден по индексу {i-k+1}")

            break

    else:

**print**("образ не найден")

else:

**print**("образ не найден")

*#!/usr/bin/env python*

*# -\*- coding: utf-8 -\*-*

*# vim:fileencoding=utf-8*

from random import shuffle

from tkinter import Canvas, Tk

BOARD\_SIZE = 4

SQUARE\_SIZE = 80

EMPTY\_SQUARE = BOARD\_SIZE \*\* 2

root = Tk()

root.title("Pythonicway Fifteen")

c = Canvas(root, width=BOARD\_SIZE \* SQUARE\_SIZE,

           height=BOARD\_SIZE \* SQUARE\_SIZE, bg='#808080')

c.pack()

def **get\_inv\_count**():

    """ Функция считающая количество перемещений """

    inversions = 0

    inversion\_board = board[:]

    inversion\_board.remove(EMPTY\_SQUARE)

    for i in **range**(**len**(inversion\_board)):

        first\_item = inversion\_board[i]

        for j in **range**(i+1, **len**(inversion\_board)):

            second\_item = inversion\_board[j]

            if first\_item > second\_item:

                inversions += 1

    return inversions

def **is\_solvable**():

    """ Функция определяющая имеет ли головоломка рещение """

    num\_inversions = get\_inv\_count()

    if BOARD\_SIZE % 2 != 0:

        return num\_inversions % 2 == 0

    else:

        empty\_square\_row = BOARD\_SIZE - (board.index(EMPTY\_SQUARE) // BOARD\_SIZE)

        if empty\_square\_row % 2 == 0:

            return num\_inversions % 2 != 0

        else:

            return num\_inversions % 2 == 0

def **get\_empty\_neighbor**(index):

*# получаем индекс пустой клетки в списке*

    empty\_index = board.index(EMPTY\_SQUARE)

*# узнаем расстояние от пустой клетки до клетки по которой кликнули*

    abs\_value = **abs**(empty\_index - index)

*# Если пустая клетка над или под клектой на которую кликнули*

*# возвращаем индекс пустой клетки*

    if abs\_value == BOARD\_SIZE:

        return empty\_index

*# Если пустая клетка слева или справа*

    elif abs\_value == 1:

*# Проверяем, чтобы блоки были в одном ряду*

        max\_index = **max**(index, empty\_index)

        if max\_index % BOARD\_SIZE != 0:

            return empty\_index

*# Рядом с блоком не было пустого поля*

    return index

def **draw\_board**():

*# убираем все, что нарисовано на холсте*

    c.delete('all')

*# Наша задача сгруппировать пятнашки из списка в квадрат*

*# со сторонами BOARD\_SIZE x BOARD\_SIZE*

*# i и j будут координатами для каждой отдельной пятнашки*

    for i in **range**(BOARD\_SIZE):

        for j in **range**(BOARD\_SIZE):

*# получаем значение, которое мы должны будем нарисовать*

*# на квадрате*

            index = str(board[BOARD\_SIZE \* i + j])

*# если это не клетка которую мы хотим оставить пустой*

            if index != str(EMPTY\_SQUARE):

*# рисуем квадрат по заданным координатам*

                c.create\_rectangle(j \* SQUARE\_SIZE, i \* SQUARE\_SIZE,

                                   j \* SQUARE\_SIZE + SQUARE\_SIZE,

                                   i \* SQUARE\_SIZE + SQUARE\_SIZE,

                                   fill='#43ABC9',

                                   outline='#FFFFFF')

*# пишем число в центре квадрата*

                c.create\_text(j \* SQUARE\_SIZE + SQUARE\_SIZE / 2,

                              i \* SQUARE\_SIZE + SQUARE\_SIZE / 2,

                              text=index,

                              font="Arial {} italic".format(int(SQUARE\_SIZE / 4)),

                              fill='#FFFFFF')

def **show\_vactory\_plate**():

*# Рисуем черный квадрат по центру поля*

    c.create\_rectangle(SQUARE\_SIZE / 5,

                       SQUARE\_SIZE \* BOARD\_SIZE / 2 - 10 \* BOARD\_SIZE,

                       BOARD\_SIZE \* SQUARE\_SIZE - SQUARE\_SIZE / 5,

                       SQUARE\_SIZE \* BOARD\_SIZE / 2 + 10 \* BOARD\_SIZE,

                       fill='#000000',

                       outline='#FFFFFF')

*# Пишем красным текст Победа*

    c.create\_text(SQUARE\_SIZE \* BOARD\_SIZE / 2, SQUARE\_SIZE \* BOARD\_SIZE / 1.9,

                  text="ПОБЕДА!", font="Helvetica {} bold".format(int(10 \* BOARD\_SIZE)), fill='#DC143C')

def **click**(event):

*# Получаем координаты клика*

    x, y = event.x, event.y

*# Конвертируем координаты из пикселей в клеточки*

    x = x // SQUARE\_SIZE

    y = y // SQUARE\_SIZE

*# Получаем индекс в списке объекта по которому мы нажали*

    board\_index = x + (y \* BOARD\_SIZE)

*# Получаем индекс пустой клетки в списке. Эту функцию мы напишем позже*

    empty\_index = get\_empty\_neighbor(board\_index)

*# Меняем местами пустую клетку и клетку, по которой кликнули*

    board[board\_index], board[empty\_index] = board[empty\_index], board[board\_index]

*# Перерисовываем игровое поле*

    draw\_board()

c.bind('<Button-1>', click)

c.pack()

board = list(**range**(1, EMPTY\_SQUARE + 1))

correct\_board = board[:]

shuffle(board)

while not is\_solvable():

    shuffle(board)

draw\_board()

root.mainloop()

# **4. Результат работы программы**

Результат работы программы представлен на Рисунках 1-2.

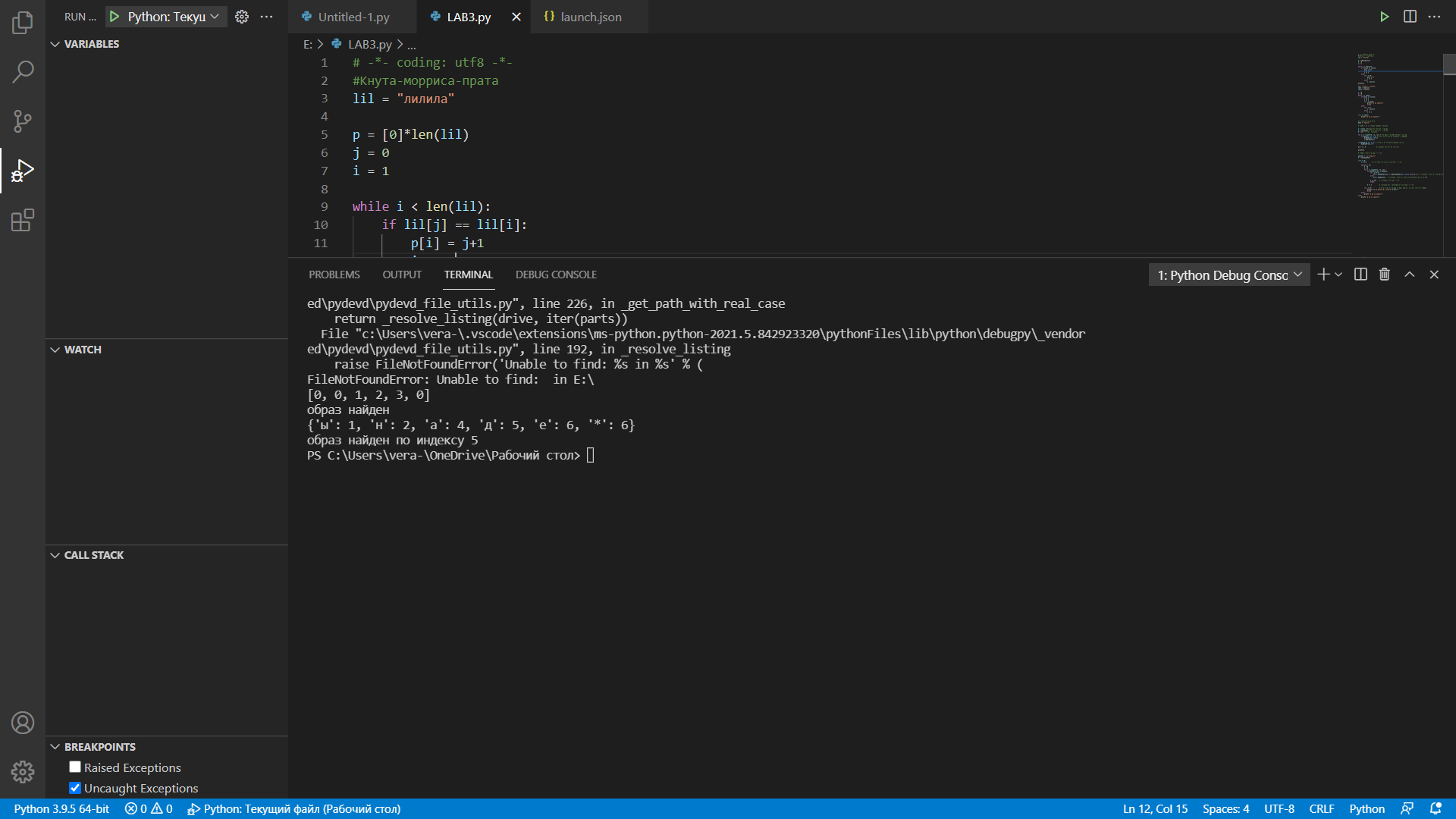
****

Рисунок 1 - Результат 1

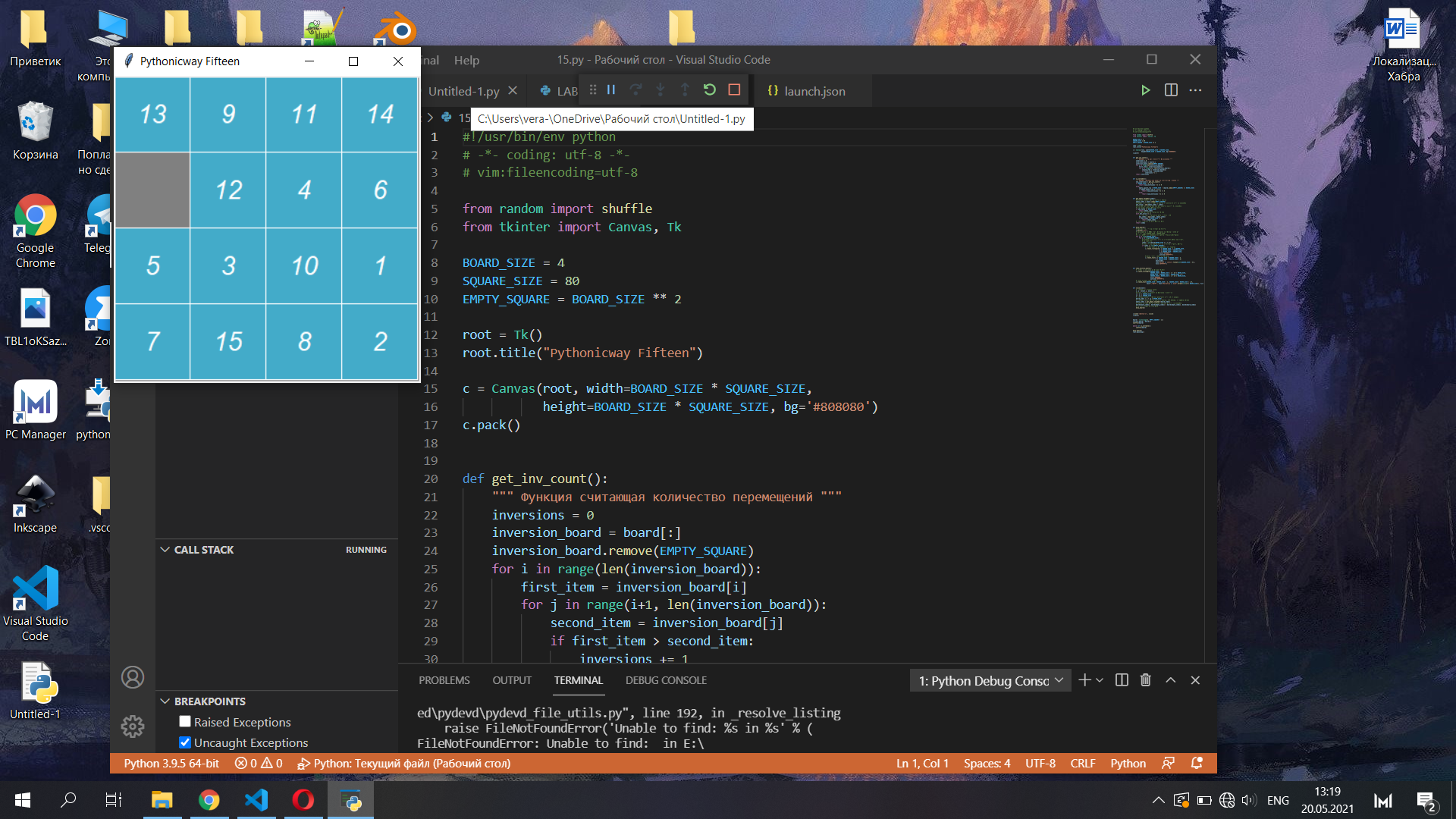
****

Рисунок 2 - Результат 2

**Вывод:** мы рассмотрели и изучили основные методы поиска подстроки в строке.

# **Список использованных источников**

1) ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчёт о научно-исследовательской работе. Структура и правила оформления

2) ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления