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

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 сим
вол 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 символ
ы 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_inde
x]
    # 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.

```
[0, 0, 1, 2, 3, 0]
образ найден
{'ы': 1, 'н': 2, 'а': 4, 'д': 5, 'е': 6, '*': 6}
образ найден по индексу 5
```

Рисунок 1 - Результат 1

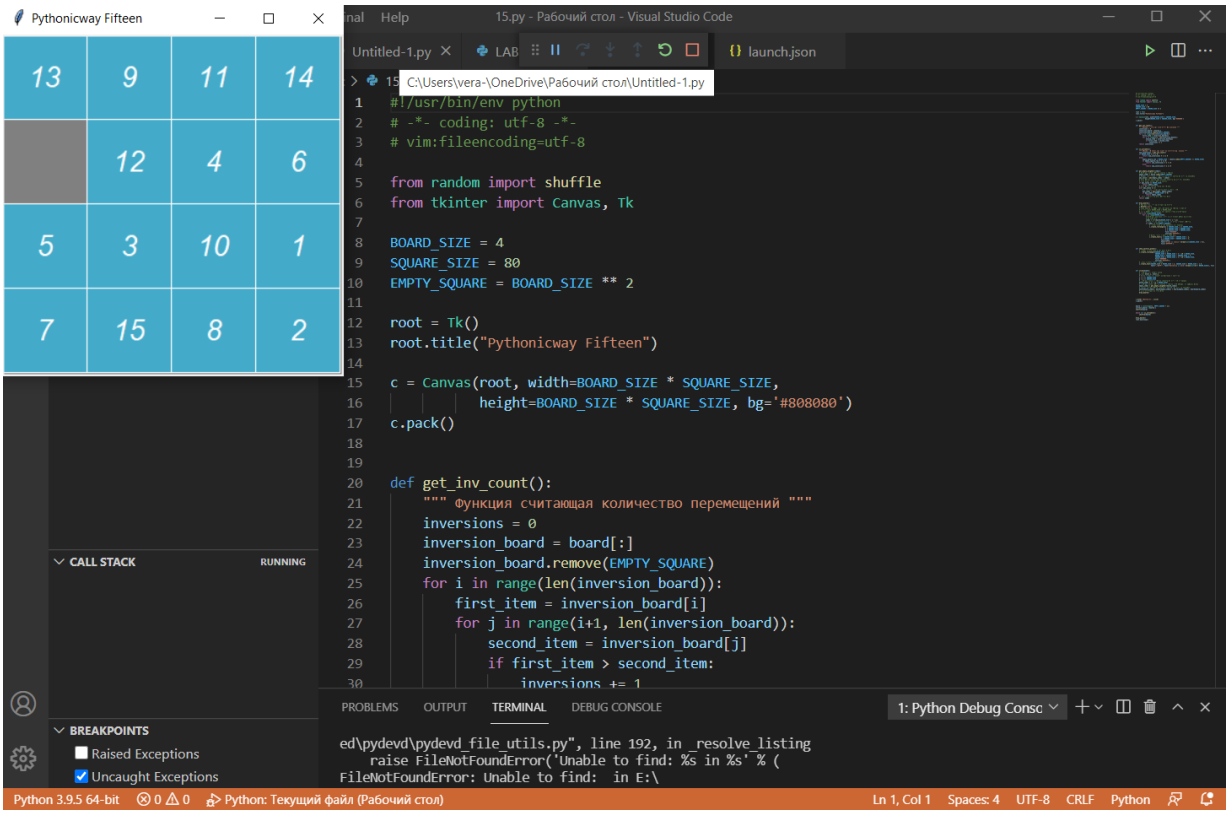

Рисунок 2 - Результат 2

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