

Федеральное агентство связи  
Ордена Трудового Красного Знамени федеральное государственное  
бюджетное образовательное учреждение высшего образования  
«Московский технический университет связи и информатики»

Кафедра «Информатика»

Лабораторная работа №3  
по дисциплине «Структура и алгоритмы обработки данных»  
«Методы поиска подстроки в строке»

Выполнил студент  
группы БФИ1902  
Кочеринский Н.В.  
Проверил: Мкртчян Г.М.

Москва 2021

# Оглавление

1 Задание на лабораторную работу.....	3
2 Решение лабораторной работы.....	3
2.1 Задание 1.....	3
2.2 Задание 2.....	6

## 1 Задание на лабораторную работу.

А) 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.

Б) Написать программу, определяющую, является ли данное расположение «решаемым», то есть можно ли из него за конечное число шагов перейти к правильному. Если это возможно, то необходимо найти хотя бы одно решение - последовательность движений, после которой числа будут расположены в правильном порядке.

## 2 Решение лабораторной работы

### 2.1 Задание 1.

Необходимо реализовать методы поиска подстроки в строки:

- 1) Кнута-Морриса-Пратта
- 2) Упрощенный Бойера-Мура

На рисунке 1 представлен результат работы программы.

*''' Метод Кнута-Морриса-Пратта'''*

```
print("\nМетод Кнута-Морриса-Пратта")
```

```
def prefixCalc(text, find):  
    new_text = find + "#" + text  
    prefix = []
```

```

for i in range(len(new_text)):
    prefix.append(0)
for i in range(1, len(new_text)):
    k = prefix[i - 1]
    while k > 0 and new_text[k] != new_text[i]:
        k = prefix[k - 1]
    if new_text[k] == new_text[i]:
        k += 1
    prefix[i] = k
return prefix

def searchKMP(text, find, ignore):
    result = []
    if ignore:
        text=text.lower()
        find=find.lower()
    prefix = prefixCalc(text, find)
    prefix = prefix[len(find) + 1:]
    for i in range(len(prefix)):
        if prefix[i]==len(find):
            result.append([i - len(find) + 1, i])
    return result

text=input("\nИсходный текст: ")
search_str=input("Подстрока: ")

print("Учитывать регистр:\n1-Выкл\n2-Вкл")
case=int(input())

if(case==1):
    result=searchKMP(text,search_str,True)
else:
    result=searchKMP(text,search_str,False)

print(result)

print("\nМетод Бойера-Мура")
def tableCalc(find):
    length=len(find)
    table = []
    for i in range(256):
        table.append(length)
    for i in range(length - 1):
        table[ord(find[i])] = length - 1 - i
    return table

```

```

def searchBM(text, find, ignore):
    result = []
    nxt = 0
    length=len(find)
    if ignore:
        text=text.lower()
        find=find.lower()
    table = tableCalc(find)
    while len(text) - nxt >= length:
        if text[nxt:(nxt + length)] == find:
            result.append((nxt, nxt + length - 1))
            nxt += table[ord(text[nxt + length - 1])]
    return result

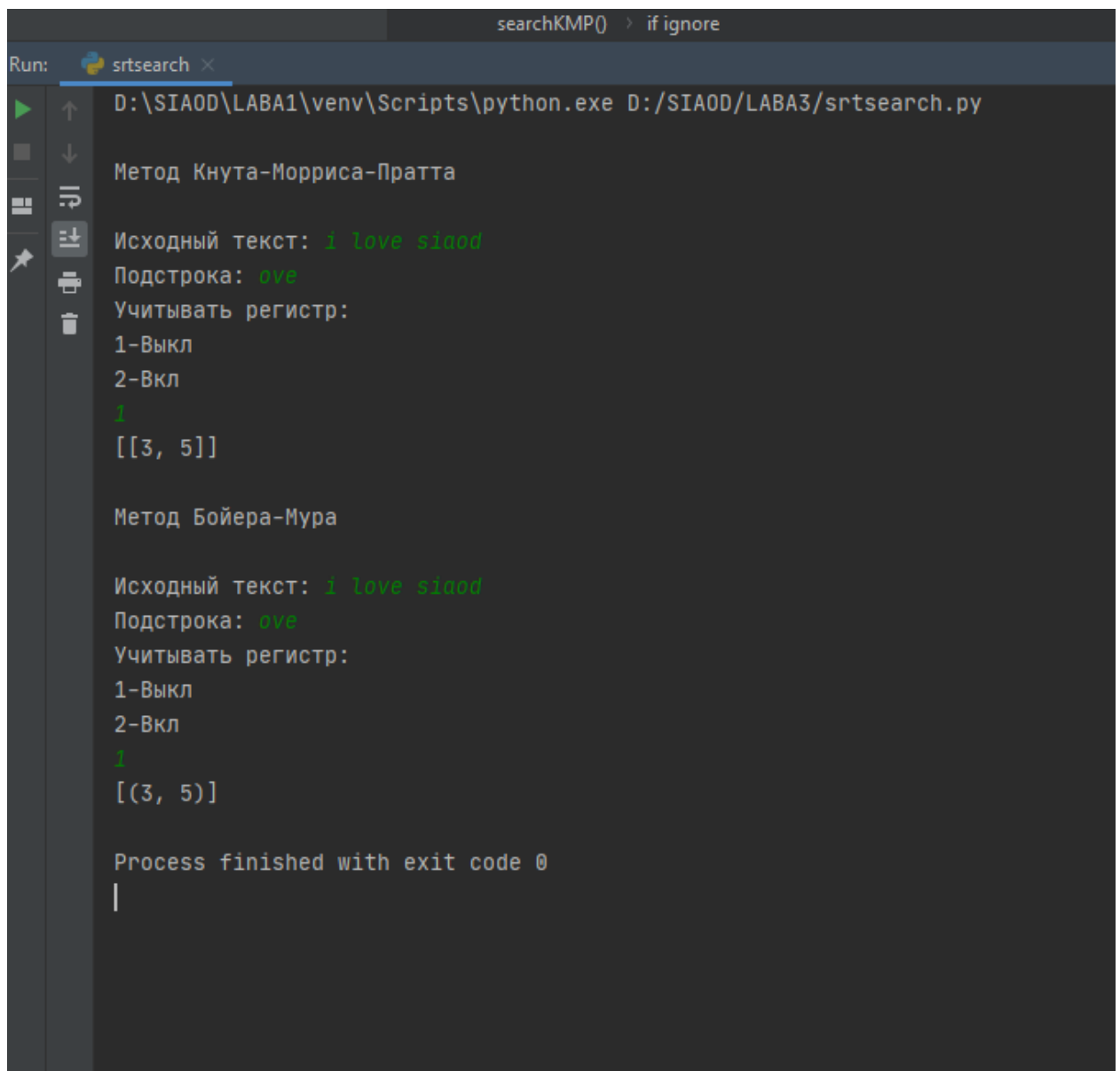
text=input("\nИсходный текст: ")
search_str=input("Подстрока: ")

print("Учитывать регистр:\n1-Выкл\n2-Вкл")
case=int(input())

if(case==1):
    result=searchBM(text,search_str,True)
else:
    result=searchBM(text,search_str,False)

print(result)

```



```
searchKMP0 > if ignore
Run: srtsearch x
D:\SIA0D\LABA1\venv\Scripts\python.exe D:/SIA0D/LABA3/srtsearch.py

Метод Кнута-Морриса-Пратта

Исходный текст: i love siaod
Подстрока: ove
Учитывать регистр:
1-Выкл
2-Вкл
1
[[3, 5]]

Метод Бойера-Мура

Исходный текст: i love siaod
Подстрока: ove
Учитывать регистр:
1-Выкл
2-Вкл
1
[(3, 5)]

Process finished with exit code 0
|
```

Рисунок 1 – Методы поиска подстроки в строке.

## 2.2 Задание 2.

Далее по плану лабораторной работы необходимо реализовать программу, определяющую, является ли данное расположение «решаемым», то есть можно ли из него за конечное число шагов перейти к правильному. Если это возможно, то необходимо найти хотя бы одно решение - последовательность движений, после которой числа будут расположены в правильном порядке.

**Входные данные:** массив чисел, представляющий собой расстановку в порядке «слева направо, сверху вниз». Число 0 обозначает пустое поле.

Например, массив [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0] представляет собой «решенную» позицию элементов.

**Выходные данные:** если решения нет, то функция должна вернуть пустой массив []. Если решение есть, то необходимо представить решение — для каждого шага записывается номер передвигаемого на данном шаге элемента.

На рисунке 2 представлен результат работы программы.

```
from queue import PriorityQueue
```

```
N = 4
```

```
# Движение пятнашек
```

```
def moves(position):
```

```
    blank = position.index(0)
```

```
    i, j = divmod(blank, N)
```

```
    offsets = []
```

```
    if i > 0: offsets.append(-N) # вниз
```

```
    if i < N - 1: offsets.append(N) # вверх
```

```
    if j > 0: offsets.append(-1) # вправо
```

```
    if j < N - 1: offsets.append(1) # влево
```

```
    for offset in offsets:
```

```
        swap = blank + offset
```

```
        yield tuple(
```

```
            position[swap] if x == blank else position[blank] if x == swap else  
            position[x] for x in range(N * N))
```

```
# Функция для определения есть решение или нет
```

```
def parity(permutation):
```

```
    seen, cycles = set(), 0
```

```
    for i in permutation:
```

```
        if i not in seen:
```

```
            cycles += 1
```

```
            while i not in seen:
```

```
                seen.add(i)
```

```
                i = permutation[i]
```

```
    return (cycles + len(permutation)) % 2
```

```
# Класс позиции
```

```

class Position:
    # Конструктор класса, который принимает позицию и начальную
    # дистанцию
    def __init__(self, position, start_distance):
        self.position = position
        self.start_distance = start_distance

    # Метод, который срабатывает при сравнении (<) объекта с другим
    # объектом
    def __lt__(self, other):
        return self.start_distance < other.start_distance

    # Метод, который срабатывает при использовании объекта как строки
    def __str__(self):
        return '\n'.join((N * '{:3}').format(*[i % (N * N) for i in self.position[i:]]) for i
        in range(0, N * N, N))

# Разгадка
SOLVED = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0)
# Загадка
start = [1, 2, 3, 4, 5, 6, 7, 8, 13, 9, 11, 12, 10, 14, 15, 0]

# Смотрим, можно ли в данной расстановке найти решение
# Если нет, то сообщаем об этом
if parity(start) == 0:
    print('Нерешаемо')
# Иначе ищем этот путь
else:

    start = tuple(start)

    # Первоначальная позиция
    p = Position(start, 0)
    print("Первоначальная позиция: " + "\n", p)
    print()

    # 1) Кладем в очередь с приоритетом первоначальную позицию
    candidates = PriorityQueue()
    candidates.put(p)

    # Кортеж посещенных позиций
    visited = set([p])

    # Откуда пришли

```



```

came_from = {p.position: None}

# Пока решение не найдено
while p.position != SOLVED:
    # 2) Извлекаем из очереди позицию с наименьшим приоритетом
    p = candidates.get()
    # 3) Кладем в очередь все соседние позиции
    # 4) Повторяем пункты 2-4 пока в пункте 2 не вытащим конечную
позицию
    for k in moves(p.position):
        if k not in visited:
            # В candidates хранятся всевозможные позиции
            candidates.put(Position(k, p.start_distance + 1))
            came_from[k] = p
            visited.add(k)

# path - последовательное решение головоломки (путь)
path = []
# Сохраняем конечную позицию
prev = p
# Идем в обратном порядке и запоминаем очередность хода в path
while p.position != start:
    # Запоминаем откуда ход
    p = came_from[p.position]
    number = p.position[prev.position.index(0)]
    path.append(number)
    prev = p
path.reverse()

print("Оптимальный путь к решению:" + "\n", path)

```

```
11         if i > 0: offsets.append(-N) # ВНИЗ
12         if i < M - 1: offsets.append(M) # ВВЕРХ
```

Run: pyat x

D:\SIA0D\LABA1\venv\Scripts\python.exe D:/SIA0D/LABA3/pyat.py

Первоначальная позиция:

1	2	3	4
5	6	7	8
13	9	11	12
10	14	15	0

Оптимальный путь к решению:

[15, 14, 10, 13, 9, 10, 14, 15]

Process finished with exit code 0

Рисунок 2 - Пятнашки

Вывод: в данной лабораторной работы были изучены и применены на практике методы поиска подстроки в строке.