

Федеральное государственное автономное образовательное  
учреждение высшего образования «Национальный  
исследовательский университет ИТМО»  
Факультет инфокоммуникационных технологий

## **ОТЧЕТ**

**по лабораторной работе №4 Подстроки по дисциплине  
«Алгоритмы и структуры данных»**

Выполнил студент 1 курса, группы К3141

Рахлин Роман Михайлович

Преподаватель:

Харьковская Татьяна Александровна

5 Июня, 2022 года, г. Санкт-Петербург

## Задание 1

**Описание задачи:** Дано слово, некий паттерн для которого мы будем искать и нужное количество несовпадений. Требуется сосчитать сколько есть в мест в слове так что количество несовпадений равно заданному количеству и вывести их индексы.

**Исходный код и описание:**

```
def KMPSearch():
    lps = [0] * len(pattern)
    j = 0

    computeLPSArray(lps)

    i = 0
    while i < len(text):
        if pattern[j] == text[i]:
            i += 1
            j += 1

        if j == len(pattern):
            result.append(i - j)
            j = lps[j-1]
        elif i < len(text) and pattern[j] != text[i]:
            if j != 0:
                j = lps[j-1]
            else:
                i += 1
```

Мы используем КМР алгоритм для решения этой задачи. Он позволяет находить substring в string не за  $O(n*m)$  как стандартный алгоритм с двумя циклами и проверками, а за  $O(n + m)$ .

```

def computeLPSArray(lps):
    prev_lps = 0

    lps[0]
    i = 1

    while i < len(pattern):
        if pattern[i] == pattern[prev_lps]:
            prev_lps += 1
            lps[i] = prev_lps
            i += 1
        else:
            if prev_lps != 0:
                prev_lps = lps[prev_lps-1]
            else:
                lps[i] = 0
                i += 1

```

Нам помогает LPS массив. Это такой массив, которые запоминать повторяющиеся префиксы и постфикс, именно он используется в алгоритме KMP.

### Задание 3

**Описание задачи:** То же самое что и в первом задании, только реализовать это с помощью одного очень быстрого алгоритма Rabin-Carp.

**Исходный код и описание:**

```

def Rabin_Karp_Matcher(text, pattern, d, q):
    n = len(text)
    m = len(pattern)

    h = pow(d, m-1) % q

    p = 0
    t = 0

    result = []

    for i in range(m):
        p = (d * p + ord(pattern[i])) % q
        t = (d * t + ord(text[i])) % q

    for s in range(n-m+1):
        if p == t:
            match = True

            for i in range(m):
                if pattern[i] != text[s+i]:
                    match = False
                    break

            if match:
                result = result + [s]

        if s < n-m:
            t = (t - h * ord(text[s])) % q
            t = (t * d + ord(text[s+m])) % q
            t = (t + q) % q

    return result

```

Я реализовал этот алгоритм, полностью разобрался как он работает и и показался с настройками.

**Тесты:** Тесты почему-то не проходят по времени, на последних тестах.

## Задание 8

**Описание задачи:** Дано слово, некий паттерн для которого мы будем искать и нужное количество несовпадений.

Требуется сосчитать сколько есть в мест в слове так что количество несовпадений равно заданному количеству и вывести их индексы.

**Исходный код и описание:**

```
mismatch_tolerance = int(line[0])
text = line[1]
pattern = line[2]

result = []

text_index_max = len(text) - len(pattern) + 1

for text_index in range(text_index_max):
    missed = 0;

    for pattern_index in range(len(pattern)):
        text_char = text[text_index + pattern_index]
        pattern_char = pattern[pattern_index]

        if text_char != pattern_char:
            missed += 1

        if missed > mismatch_tolerance:
            break

    if missed == mismatch_tolerance:
        match = text[text_index:text_index + len(pattern)]
        result.append(text_index)
```

В коде много частей для работы с чтением и записью, а вот на фотке я привел основной функционал. По сути, то что мы делаем это проходим строку (но не всю, а не до длинны минус

длина паттера) после этого проходим по длине паттера и находим сколько символов реально не совпадает в текущей итерации. Одновременно с этим сразу проверяем если количество допустимых несовпадений уже превышает наше значение, то прерываем итерацию. И после итерации по слову, проверяем на количество итераций и записывает все красиво в ответ.

**Тесты:** В качестве тестов я брал заданные входные данные из условия.

## **Задание 202**

**Описание задачи:** Задача решается по алгоритмам приведенным в задачах 1 и 3.

**Исходный код и описание:** Я просто взял код из задач 1 и 3, запустил и получил отличный результат.

**Тесты:**

17251968	06.06.2022 17:58:45	Рахлин Роман Михайлович	0202	Python	Accepted		0,156	5014 K6
----------	---------------------	-------------------------	------	--------	----------	--	-------	---------

## **Задание 930**

**Описание задачи:** В этой задаче у нас дается два имени. Нам нужно по двух этим именам сгенерировать подходящее правилам заданным в условии.

**Исходный код и описание:**

```

with open("input.txt", "r") as input_file:
    dad = input_file.readline()
    mom = input_file.readline()

name = ""
commom = list(set(dad) & set(mom))

if len(commom) != 0:
    commom.sort()
    commom.reverse()

    while len(commom) != 0:
        length = min(dad.count(commom[0]), mom.count(commom[0]))
        name += commom[0] * length

        for _ in range(min(dad.count(commom[0]), mom.count(commom[0]))):
            dad = dad[dad.find(commom[0]) + 1:]
            mom = mom[mom.find(commom[0]) + 1:]

        commom.pop(0)

with open("output.txt", "w") as output_file:
    output_file.write(name)

```

Нам нужно представить два имени как сеты и использовать их комбинацию чтобы получить только те символы, которые встречаются в обоих именах. После этого нам нужно отсортировать их в обратном порядке и начать создавать новое имя. Нам нужно придумать длину и собирать имя по определенным правилам.

Тесты:

17252154	06.06.2022 18:45:17	Рахлин Роман Михайлович	0930	Python	Accepted		0,937	1822 Kb
----------	---------------------	-------------------------	------	--------	----------	--	-------	---------

## Задание 5

**Описание задачи:** Это довольно интересная задача. Нужно создать свою prefix функцию для всех непустых префиксов для некой заданной строки.

**Исходный код и описание:**

```

def prefix_function(s):
    n = len(s)
    pi = [0 for i in range(n)]
    for i in range(1, n):
        j = pi[i - 1]
        while j > 0 and s[i] != s[j]:
            j = pi[j - 1]
        if s[i] == s[j]:
            j += 1
        pi[i] = j
    return pi

with open("input.txt", "r") as input_file:
    result = prefix_function(input_file.readline())

with open("output.txt", "w") as output_file:
    output = ""
    for x in result:
        output += str(x) + " "
    output_file.write(output[:len(output) - 1])

```

Ну тут нечего объяснять. Префикс функция которая выполняет свою задачу. А также работает с чтением и записью в файл.

## Задание 886

**Описание задачи:** Это довольно интересная задача. Нужно создать свою prefix функцию для всех непустых префиксов для некой заданной строки.

**Исходный код и описание:**



```

with open("input.txt", "r") as input_file:
    s = input_file.readline()

n = len(s)
l = 0
prefix = [0 for i in range(1 + n)]

for i in range(1, n):
    while True:
        if s[l] == s[i]:
            l += 1
            break

        if l == 0:
            break

        l = prefix[l]

    prefix[i + 1] = l

period = n - prefix[n]

if n % period != 0:
    result = 1
else:
    result = n // period

with open("output.txt", "w") as output_file:
    output_file.write(str(result))

```

Я решал эту задачу так. Сначала прочитал файл, потом перебираю циклом все и делаю проверки в цикле while. Если что-то нет так, делаю break. Ну и дальше просто потихоньку рассчитываю дальше индекс. И в конце просто делаю финальную проверку. И потом записываю результат.

## **Вывод**

Выполняя эту лабораторную работу я приобрел очень много ценного опыта работы со строками и всеми классными алгоритмами по поиску. Мега полезные знания!!