

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ  
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №4  
по курсу «Алгоритмы и структуры данных»

Тема: Подстроки

Вариант 2

Выполнил:

Вилис З.М. (фамилия имя)

К3244 (номер группы)

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

## Содержание отчета

<b>Содержание отчета</b>	2
<b>Задачи по варианту 3</b>	
Задание №1. Наивный поиск подстроки в строке[1 балл]	3
Задание №5. Префикс-функция[1.5 балл]	4
Задание №7. Наибольшая общая подстрока[2 балла]	6
<b>Дополнительные задачи 8</b>	
Задание №5+. Наивный поиск подстроки в строке[2 балла]	8
Задание №6. Z-функция[1.5 балла]	11
Задание №8+. Подстроки из одинаковых букв[2 балла]	13
Задание №6+. Сдвиг текста[2 балла]	14
Задание №1+. Последнее слово Джека[2 балла]	14

## Задачи по варианту

### Задание №1. Наивный поиск подстроки в строке[1 балл]

Даны строки  $p$  и  $t$ . Требуется найти все вхождения строки  $p$  в строку  $t$  в качестве подстроки.

```
f = open('input.txt')
t2 = f.readline()
t1 = t2[:len(t2)-1]
p1 = f.readline()

def naiveStringMatcher(t, p):
    n = len(t)
    m = len(p)
    ans = []
    for i in range(n-m+1):
        if t[i:i+m] == p:
            ans.append(i+1)
    return ans

z = open('output.txt', 'w')
itog = naiveStringMatcher(p1, t1)
z.write(str(len(itog)) + '\n')
itog.sort()
for i in itog:
    z.write(str(i) + ' ')
```

Текстовое объяснение решения:

Функция `naiveStringMatcher` принимает на вход искомую подстроку и строку, в которой проводится поиск. Переменным  $n$  и  $m$  присваиваются длины полной строки и искомой. Заводится пустой массив, в него в течении цикла, рассматривающего все подстроки длины  $m$  записываются индексы начала найденных строк.

Результат работы кода на примерах из задачи:

aba	2
abaCaba	1 5

Нижняя граница значений:

a	1
a	1

Верхняя граница значений:

51	wr
35 125 994	wigggyueoqf

	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.0004761219024658203	22.8
Пример из задачи	0.0008678436279296875	23.2
Пример из задачи	0.0008130073547363281	23.2
Верхняя граница диапазона значений входных данных из текста задачи(10000 элементов)	0.0019190311431884766	23.2

Вывод по задаче: задача реализует наивный алгоритм поиска подстроки в строке.

### Задание №5. Префикс-функция[1.5 балл]

Постройте префикс-функцию для всех непустых префиксов заданной строки s.

```
f = open('input.txt')
st = f.readline()

def prefixFunction(s):
    p = [0] * len(s)
    for i in range(1, len(s)):
        k = p[i-1]
        while k > 0 and s[i] != s[k]:
            k = p[k - 1]
        if s[i] == s[k]:
            p[i] = k + 1
```

```

        k += 1
        p[i] = k
    return p


z = open('output.txt', 'w')
itog = prefixFunction(st)
for i in itog:
    z.write(str(i) + ' ')

```

Текстовое объяснение решения:

Функция `prefixFunction` принимает на вход строку `s`. Создается массив из 0 длины `s`. Запускается цикл проходящий по всей строке. В `k` записывается элемент предыдущий текущему рассматриваемому. Пока это значение больше нуля и элемент строки рассматриваемый не равен индексу, равному `k`. Если они равны, то `k` сдвигается на 1. Далее в массив на `i` тое место записывается получившееся `k`.

Результат работы кода на примерах из задачи:

aaaAAA	0	1	2	0	0	0	
abacaba	0	0	1	0	1	2	3
							

Нижняя граница значений:

а	0

Верхняя граница значений:

raatqr gjuaoui	0	0	0	0	0	1	0
sbjomusrogmyu'	0	0	0	0	0	0	0

	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.000537236892368276	22.8
Пример из задачи	0.000892382378927393	23.2
Пример из задачи	0.0008130073547363281	23.2
Верхняя граница диапазона значений входных данных из текста задачи(1000 элементов)	0.008283726372638723	23.2

Вывод по задаче: Задача реализует префикс-функцию( ищет для каждой текущей подстроки длины  $n$ , постоянно увеличивающейся на 1, начинающейся с 0-го элемента рассматриваемой строки, максимальную длину совпадающих суффикса и префикса)

#### **Задание №7. Наибольшая общая подстрока[2 балла]**

В задаче на наибольшую общую подстроку даются две строки  $s$  и  $t$ , и цель состоит в том, чтобы найти строку  $w$  максимальной длины, которая является подстрокой как  $s$ , так и  $t$ . Это естественная мера сходства между двумя строками. Задача имеет применения для сравнения и сжатия текстов, а также в биоинформатике. Эту проблему можно рассматривать как частный случай проблемы расстояния редактирования (Левенштейна), где разрешены только вставки и удаления. Следовательно, ее можно решить за время  $O(|s||t|)$  с помощью динамического программирования. Есть также весьма нетривиальные структуры данных для решения этой задачи за линейное время  $O(|s| + |t|)$ . В этой задаче ваша цель – использовать хеширование для решения почти за линейное время.

```
def longest_common_subsequence(s1, s2):
    dp = [[0] * (len(s2)+1) for _ in range(len(s1)+1)]
    max_len = 0
    start_index = 0
    st_ind = 0
    for i in range(1, len(s1)+1):
        for j in range(1, len(s2)+1):
            if s1[i-1] == s2[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
                if dp[i][j] > max_len:
                    max_len = dp[i][j]
                    start_index = i - max_len
```

```

        st_ind = j - max_len
    else:
        dp[i][j] = 0

    return max_len, start_index, st_ind

f = open('input.txt')
s1, s2 = f.readline().split()
length, start_index, s_i = longest_common_subsequence(s1, s2)
z = open('output.txt', 'w')
z.write(str(start_index) + ' ' + str(s_i) + ' ' + str(length))

```

Текстовое объяснение решения:

Создается матрица `dp` элементы которой заполняются по следующему принципу:

Если символы `s1[i-1]` и `s2[j-1]` совпадают то элемент `dp[i][j]` равен `dp[i-1][j-1] + 1`. Если он максимальный то становится сам максимальным и также записываются стартовые индексы. Иначе равен нулю.

Результат работы кода на примерах из задачи:

cool toolbox	1 1 3
aaa bb	0 0
aabaa babbaab	0 4 3

	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.0035456576576465563	22.8
Пример из задачи	0.0032543654563453645	23.2

Пример из задачи	0.00545435465465344	23.2
Верхняя граница диапазона значений входных данных из текста задачи(1000 элементов)	8.5345354564654654453	23.2

Вывод по задаче: Задача решается при помощи хэширования

## Дополнительные задачи

### Задание №5+. Наивный поиск подстроки в строке[2 балла]

Найти все вхождения строки T в строке S.

#### Входные данные

В первой строке входного файла INPUT.TXT записана строка S, во второй строке записана строка T. Обе строки состоят только из английских букв. Длины строк могут быть в диапазоне от 1 до 50 000 включительно.

#### Выходные данные

В выходной файл OUTPUT.TXT нужно вывести все вхождения строки T в строку S в порядке возрастания. Нумерация позиций строк начинается с нуля.

```
n = input()
s = input()
res = []
for i in range(len(n) - len(s) + 1):
    if n[i:].startswith(s):
        res.append(i)
print(*sorted(res))
```

Текстовое объяснение решения:

В цикле проверяется пошагово будет ли содержаться подстрока в строке.



Тест	Результат	Время	Память
1	Accepted	0,015	462 Kб
2	Accepted	0,015	466 Kб
3	Accepted	0,015	466 Kб
4	Accepted	0,015	458 Kб
5	Accepted	0,015	470 Kб
6	Accepted	0,015	462 Kб
7	Accepted	0,093	590 Kб
8	Accepted	0,078	558 Kб
9	Accepted	0,078	586 Kб
10	Accepted	0,031	706 Kб
11	Time limit exceeded	0,218	2262 Kб

Вывод по задаче: Задача не проходит на больших значениях, так как возможности питона ограничены.

#### Задание №6. Z-функция[1.5 балла]

Постройте Z-функцию для заданной строки s.

```
def zfunction(s):
    zf = [0] * len(s)
    left, right = 0, 0
    for i in range(1, len(s)):
        zf[i] = max(0, min(right - i, zf[i - left]))
        while i + zf[i] < len(s) and s[zf[i]] == s[i + zf[i]]:
            zf[i] = zf[i] + 1
        if i + zf[i] > right:
            left = i
            right = i + zf[i]
    return zf

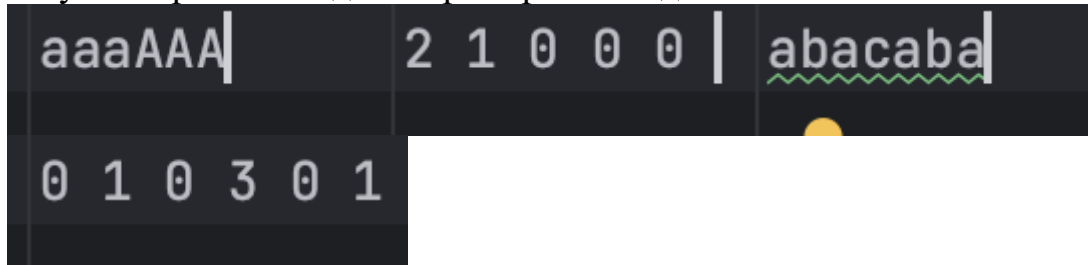
f = open('input.txt')
stroka1 = f.readline()
stroka = stroka1[:len(stroka1)]
rez = zfunction(stroka)
with open('output.txt', 'w') as z:
    for i in range(1, len(rez)):
        z.write(str(rez[i]) + ' ')
```

Текстовое объяснение решения:

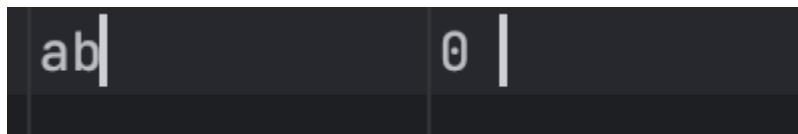
Код реализует z-функцию, получающую на вход строку и возвращающую массив z, такой что каждый i-й элемент равен длине максимальной

подстроки начинающейся с  $i$  того элемента которая равна префиксу строки. Происходит пробег по всем элементам строки  $s$ , начиная с 1. Чтобы высчитать  $zf[i]$  берется минимум между длиной от правого конца до числа и  $zf$  с индексом от  $i$  до левого края. Далее происходит перебор в котором если дальнейшее значение равняется, число в массиве увеличивается на 1. Если переходит границу, то левая граница сдвигается на рассматриваемый элемент, а правая на элемент плюс найденная длина префикса.

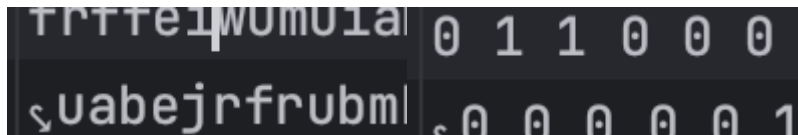
Результат работы кода на примерах из задачи:



Минимальное значение:



Максимальное значение:



	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.0456237236736273	22.9
Пример из задачи	0.06374672343476343	23.1
Пример из задачи	0.063476736478347637	23.1
Верхняя граница диапазона значений входных данных из текста задачи(1000 элементов)	1.384838432949343981	23.2

Вывод по задаче: Данная задача позволяет реализовывать z-функцию

### Задание №8+. Подстроки из одинаковых букв[2 балла]

В заданной строке, состоящей из малых английских букв, необходимо найти пару самых длинных подстрок, состоящих из одних и тех же букв (возможно, в разном порядке). Например, в строке twotwow это будут подстроки wotwo и otwow.

#### Входные данные

Входной файл INPUT.TXT содержит исходную строку, длиной от 1 до 100 символов.

#### Выходные данные

Выходной файл OUTPUT.TXT должен содержать единственное число – длину подстрок в максимальной паре, или 0, если таких подстрок в строке нет.

```
s = input()
s1 = sorted(s)
if s1 == list(set(s1)):
    print(0)
else:
    otvet = 0
    for i in range(len(s)):
        for j in range(len(s) - 1, -1, -1):
            if s[i] == s[j]:
                jj = j
                break
            sum = jj - i
            if sum > otvet:
                otvet = sum
    print(otvet)
```

Текстовое объяснение решения:

Строка вводится и сортируется по возрастанию по кодам букв. Если этот список равен списку из множества значений строки, то это означает, что повторяющихся значений нет и выводится ноль. Иначе запускается цикл, который проходит по всем элементам строки и второй проходящий с обратной стороны. Если находящийся в начале равен рассматриваемому с конца, то в переменную jj записывается j. И цикл прерывается. В конце цикла в переменную sum записывается разница между jj и i. Если sum > otvet, то в otvet записывается sum.

Тест	Результат	Время	Память
1	Accepted	0,046	518 Кб
2	Accepted	0,031	514 Кб
3	Accepted	0,046	506 Кб
4	Accepted	0,015	514 Кб
5	Accepted	0,031	506 Кб
6	Accepted	0,031	514 Кб
7	Accepted	0,015	514 Кб
8	Accepted	0,015	514 Кб
9	Accepted	0,015	510 Кб
10	Accepted	0,031	514 Кб
11	Accepted	0,015	510 Кб
12	Accepted	0,015	510 Кб
13	Accepted	0,015	514 Кб
14	Accepted	0,015	510 Кб
15	Accepted	0,031	510 Кб
16	Accepted	0,015	506 Кб
17	Accepted	0,015	518 Кб
18	Accepted	0,031	506 Кб
19	Accepted	0,031	514 Кб
20	Accepted	0,015	510 Кб

Вывод по задаче: Данная задача находит подстроки с одинаковыми буквами за короткое время.

#### Задание №6+. Сдвиг текста[2 балла]

Мальчик Кирилл написал однажды на листе бумаги строчку, состоящую из больших и маленьких английских букв, а после этого ушел играть в футбол. Когда он вернулся, то обнаружил, что его друг Дима написал под его строкой еще одну строчку такой же длины. Дима утверждает, что свою строчку он получил циклическим сдвигом строки Кирилла направо на несколько шагов (циклический сдвиг строки abcde на 2 позиции направо даст строку deabc). Однако Дима известен тем, что может случайно ошибиться в большом количестве вычислений, поэтому Кирилл в растерянности - верить ли Диме? Помогите ему!

По данным строкам выведите минимально возможный размер сдвига вправо или -1, если Дима ошибся.

```
a = input()
b = input()

if a == b:
    print(0)
    exit()

x = len(a)
for z in range(1, x):
    c = a[x - z:] + a[:x - z]
    if b == c:
        print(z)
        exit()

print(-1)
```

Текстовое объяснение решения:

Если строки равны, значит никакого сдвига не было и выводится 0. Иначе в переменную x записывается длина первой строки. Запускается цикл от 1 до этой длины. Далее в переменную c записывается строка которая могла

получиться при возможном сдвиге на  $z$  элементов. Если она действительно получилась, то выводится число.

Тест	Результат	Время	Память
1	Accepted	0,031	462 Кб
2	Accepted	0,015	458 Кб
3	Accepted	0,031	466 Кб
4	Accepted	0,015	462 Кб
5	Accepted	0,031	462 Кб
6	Accepted	0,015	462 Кб
7	Accepted	0,015	466 Кб
8	Accepted	0,015	470 Кб
9	Accepted	0,031	586 Кб
10	Accepted	0,046	862 Кб
11	Accepted	0,046	666 Кб
12	Accepted	0,031	662 Кб
13	Accepted	0,046	670 Кб
14	Accepted	0,046	662 Кб
15	Accepted	0,046	670 Кб
16	Accepted	0,046	670 Кб
17	Accepted	0,046	662 Кб
18	Accepted	0,015	466 Кб

Вывод по задаче: Задача реализуется при помощи  $z$ -функции.

### Задание №1+. Последнее слово Джека[2 балла]

Джек недавно прочитал на заборе интересное и новое для него слово. Оно настолько понравилось Джеку, что он захотел сам придумать ещё какое-нибудь интересное слово. Но только ничего у него не вышло — все придуманные им слова состояли из префиксов исходного слова и поэтому не приносили радости. Он стал придумывать всё более и более длинные слова, но ни одно из них не было оригинальным...

И вот настало время Джеку сказать своё последнее слово.

```
def z_f(s):
    zf = [0] * len(s)
    left, right = 0, 0
    for i in range(1, len(s)):
        zf[i] = max(0, min(right - i, zf[i - left]))
        while i + zf[i] < len(s) and s[zf[i]] == s[i + zf[i]]:
            zf[i] = zf[i] + 1
        if i + zf[i] > right:
            left = i
            right = i + zf[i]
    return zf

def main():
    f = input()
    s = input()
    f += '#' + s
    z = z_f(f)
    r = 0
    l = f.find('#') + 1
    v = []
    for i in range(f.find('#') + 1, len(f)):
        if z[i] == 0:
            if i > r:
                print("Yes")
```

```

        return
    else:
        if z[i] + i - 1 > r:
            r = z[i] + i - 1
            if i != l:
                v.append(f[l:i])
            l = i
    v.append(f[l:])
    print("No")
    first = s[0]
    if len(v[len(v)-1])>1 and (v[len(v)-1][len(v[len(v)-1])-1]) ==
first:
        v[len(v)-1] = v[len(v)-1][:len(v[len(v)-1])-1]
        v.append(first)
    for sub_str in v:
        print(sub_str, end=' ')
    print()

if __name__ == "__main__":
    main()

```

Текстовое объяснение решения:

Вводятся две строки, они склеиваются между собой знаком решетки. В z записывается z функция от этой строки. Задаются правая и левая границы. Запускается цикл проходящий по той части итоговой строки где находится вторая строка. Если z-функция на текущем символе равна нулю на всех значениях, то мы выводим, что строка подходит под условия. Если значение z-функции плюс номер текущего символа больше чем левая граница то левой границей становится оно само. Если номер не равен правой границе то записывается срез в список.

Результаты работы кода на примерах из задачи:

```

abracadabra
abrabracada
no
abr abracad a

```

Вывод по задаче: В очень многих задачах данной лабораторной используется z-функция.

Лабораторная работа позволила познакомиться с принципами работы с подстроками, в особенности с z-функциями.