

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

Выполнила:
Бочкарева Е.А.
К3144

Проверила:
Артамонова В.Е.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №3 Паттерн в тексте	3
Задача №4 Равенство подстрок	6
Задача №8 Шаблоны с несовпадениями	8
Дополнительные задачи	10
Задача №1 Наивный поиск подстроки в строке	10
Задача №5 Префикс-функция	12
Задача №6 Z-функция	14
Задача №7 Наибольшая общая подстрока	16

Задачи по варианту

Задача №3 Паттерн в тексте

В этой задаче ваша цель – реализовать алгоритм Рабина-Карпа для поиска заданного шаблона (паттерна) в заданном тексте.

- **Формат ввода / входного файла (input.txt).** На входе две строки: паттерн P и текст T . Требуется найти все вхождения строки P в строку T в качестве подстроки.
- **Ограничения на входные данные.** $1 \leq |P|, |T| \leq 10^6$. Паттерн и текст содержат только латинские буквы.
- **Формат вывода / выходного файла (output.txt).** В первой строке выведите число вхождений строки P в строку T . Во второй строке выведите в возрастающем порядке номера символов строки T , с которых начинаются вхождения P . Символы нумеруются с единицы.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input	output	input	output	input	output
aba	2	Test	1	aaaaa	3
abacaba	1 5	testTesttesT	5	baaaaaa	2 3 4

- В первом примере паттерн *aba* можно найти в позициях 1 (**ab**acaba) и 5 (abac**ab**a) текста *abacaba*.
Паттерн и текст в этой задаче чувствительны к регистру. Поэтому во втором примере паттерн *Test* встречается только в 45 позиции в тексте *testTesttesT*.
Обратите внимание, что вхождения шаблона в тексте могут перекрываться, и это нормально, вам все равно нужно вывести их все.
- Используйте оператор `==` в Python вместо реализации собственной функции `AreEqual` для строк, потому что встроенный оператор `==` будет работать намного быстрее.
- Проверяем **обязательно** – на [OpenEdu](#), курс Алгоритмы программирования и структуры данных, неделя 9, наблюдаемая задача.

Листинг кода:

```
import random

def check(P, T):
    alph = {}
    for i in range(len(T) - len(P)):
        if T[i] not in alph:
            alph[T[i]] = []
    for i in range(len(P)):
        if P[i] not in alph:
            return False
    return True

def PolyHash(P, p, x):
    res = 0
    for i in reversed(range(n)):
```

```

        res = (res * x + ord(P[i])) % p
    return res % p

def PrecomputeHashes(T, p, x):
    global n, m
    H = [0] * (m - n + 1)
    S = T[m - n : m]
    H[m - n] = PolyHash(S, p, x)
    y = 1
    for i in range(1, n + 1):
        y = (y * x) % p
    for i in range(m - n - 1, -1, -1):
        H[i] = (x * H[i + 1] + ord(T[i]) - y * ord(T[i + n]) + p) % p
    return H

def RabinKarp(pattern, text):
    global n, m
    p = 10 ** 9 + 9
    x = 127
    count = 0
    res = []
    hPattern = PolyHash(pattern, p, x)
    H = PrecomputeHashes(text, p, x)
    for i in range(m - n + 1):
        if hPattern != H[i]:
            continue
        # if text[i:i+n] == pattern:
        count += 1
        res.append(str(i + 1))
    with open("output.txt", "w") as f:
        f.write(str(count) + "\n" + " ".join(res))

with open("input.txt") as f:
    P = f.readline()[:-1]
    T = f.readline()
if check(P, T):
    n, m = len(P), len(T)
    RabinKarp(P, T)
else:
    with open("output.txt", "w") as f:

```

```
f.write("0")
```

Пояснение к решению:

Этот код реализует алгоритм поиска подстроки в строке с использованием алгоритма Рабина-Карпа. Алгоритм эффективно находит все вхождения подстроки (шаблона) в строку, применяя хеширование для быстрого сравнения.

Результат работы на примерах из текста к задаче:

input.txt	output.txt
1 aba	1 2
2 abacaba	2 1 5

input.txt	output.txt
1 Test	1 1
2 testTesttesT	2 5

input.txt	output.txt
1 aaaaaa	1 3
2 baaaaaaa	2 2 3 4

Вывод по задаче:

Смогла реализовать алгоритм Рабина-Карпа.

Задача №4 Равенство подстрок

В этой задаче вы будете использовать хеширование для разработки алгоритма, способного предварительно обработать заданную строку s , чтобы ответить эффективно на любой запрос типа «равны ли эти две подстроки s ?» Это, в свою очередь, является основной частью во многих алгоритмах обработки строк.

- **Формат ввода / входного файла (input.txt).** Первая строка содержит строку s , состоящую из строчных латинских букв. Вторая строка содержит количество запросов q . Каждая из следующих q строк задает запрос тремя целыми числами a , b и l .
- **Ограничения на входные данные.** $1 \leq |s| \leq 500000$, $1 \leq q \leq 100000$, $0 \leq a, b \leq |s| - l$ (следовательно, индексы a и b начинаются с 0).
- **Формат вывода / выходного файла (output.txt).** Для каждого запроса выведите «Yes», если подстроки $s_a s_{a+1} \dots s_{a+l-1} = s_b s_{b+1} \dots s_{b+l-1}$ равны, и «No» – если не равны.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб.
- Пример:

input	output
trololo	Yes
4	Yes
0 0 7	Yes
2 4 3	No
3 5 1	
1 3 2	

Листинг кода:

```
import random

def PolyHash(P, l, p, x):
    res = 0
    for i in reversed(range(l)):
        res = (res * x + ord(P[i])) % p
    return res % p

def PrecomputeHashes(T, l, k, p, x):
    H = [0] * (l - k + 1)
    S = T[l - k : l]
    H[l - k] = PolyHash(S, k, p, x)
    y = 1
    for i in range(1, k + 1):
        y = (y * x) % p
    for i in range(l - k - 1, -1, -1):
        H[i] = (x * H[i + 1] + ord(T[i]) - y * ord(T[i + k]) + p) % p
    return H

with open('input.txt') as f:
    with open('output.txt', 'w') as f1:
        s = f.readline().strip()
        lS = len(s)
```

```

q = int(f.readline())
p = 10**9+7
x = random.randint(1, p-1)
for i in range(q):
    a, b, l = map(int, f.readline().split())
    H = PrecomputeHashes(s, lS, l, p, x)
    if H[a] == H[b]:
        f1.write('Yes\n')
    else:
        f1.write('No\n')

```

Пояснение к решению:

Этот код реализует проверку того, являются ли два подстрочных фрагмента строки идентичными, с помощью алгоритма хеширования подстрок (используя многочленное хеширование). Данный подход позволяет эффективно сравнивать подстроки с использованием предварительно вычисленных хешей.

Результат работы на примерах из текста к задаче:

input.txt		output.txt	
1	trololo	1	Yes
2	4	2	Yes
3	0 0 7	3	Yes
4	2 4 3	4	No
5	3 5 1		
6	1 3 2		

Вывод по задаче:

В данной задаче я научилась эффективно определять равенство подстрок.

Задача №8 Шаблоны с несовпадениями

Естественным обобщением задачи сопоставления паттернов, текстов является следующее: найти все места в тексте, расстояние (различие) от которых до образца достаточно мало. Эта проблема находит применение в текстовом поиске (где несовпадения соответствуют опечаткам) и биоинформатике (где несовпадения соответствуют мутациям).

В этой задаче нужно решить следующее. Для целочисленного параметра k и двух строк $t = t_0t_1\dots t_{m-1}$ и $p = p_0p_1\dots p_{n-1}$, мы говорим, что p встречается в t в знаке индекса i с не более чем k несовпадениями, если строки p и $t[i:i+p] = t_it_{i+1}\dots t_{i+n-1}$ различаются не более чем на k знаков.

- **Формат ввода / входного файла (input.txt).** Каждая строка входных данных содержит целое число k и две строки t и p , состоящие из строчных латинских букв.
- **Ограничения на входные данные.** $0 \leq k \leq 5$, $1 \leq |t| \leq 200000$, $1 \leq |p| \leq \min |t|, 100000$. Суммарная длина строчек t не превышает 200 000, общая длина всех p не превышает 100 000.
- **Формат вывода / выходного файла (output.txt).** Для каждой тройки (k, t, p) найдите все позиции $0 \leq i_1 < i_2 < \dots < i_l < |t|$ в которых строка p встречается в строке t с не более чем k несоответствиями. Выведите l и i_1, i_2, \dots, i_l .
- Ограничение по времени. 40 сек. (Python), 2 сек (C++).
- Ограничение по памяти. 512 мб.
- Пример:

input	output
0 ababab baaa	0
1 ababab baaa	1 1
1 xabcabc ccc	0
2 xabcabc ccc	4 1 2 3 4
3 aaa xxx	1 0

- **Объяснение:**
Для первой тройки точных совпадений нет. Для второй тройки baaa находится на расстоянии один от паттерна с началом в индексе 1 *ababab*. Для третьей тройки нет вхождений не более чем с одним несовпадением. Для четвертой тройки любая (длина три) подстрока p , содержащая хотя бы одну букву c , находится на расстоянии не более двух от t . Для пятой тройки t и p различаются тремя позициями.
- Начните с вычисления хеш-значений префиксов t и p и их частичных сумм. Это позволяет сравнивать любые две подстроки t и p за ожидаемое постоянное время. Для каждой позиции-кандидата i выполните k шагов вида «найти следующее несоответствие». Каждое такое несоответствие можно найти с помощью бинарного поиска.

Листинг кода:

```
import sys

def pos(t, p, k):
    s = p + t
    n = len(s)
    z = [[0] * (k + 1) for i in range(n)]
    l = r = 0
    for i in range(1, n):
        if i <= r:
            z[i][0] = min(z[i - 1][0], r - i + 1)
            while i + z[i][0] < n and s[i + z[i][0]] == s[z[i][0]]:
                z[i][0] += 1
            if i + z[i][0] > r:
                l, r = i, i + z[i][0] - 1
        for j in range(1, k + 1):
```



```

        z[i][j] = min(z[i][j - 1] + 1, n - i)
        while i + z[i][j] < n and s[i + z[i][j]] == s[z[i][j]]:
            z[i][j] += 1
        if z[i][k] >= len(p) and i >= len(p):
            yield i - len(p)

def main():
    sys.stdin = open('input.txt', "r")
    result = ""
    for line in sys.stdin:
        k, t, p = line.split()
        k = int(k)
        ans = list(pos(t, p, k))
        result += str(len(ans)) + " " + " ".join(map(str, ans)) + "\n"

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

if __name__ == '__main__':
    main()

```

Пояснение к решению:

Этот код решает задачу поиска всех вхождений подстроки **p** в строку **t** с использованием модифицированного Z-алгоритма, который учитывает до **k** отличий (ошибок) между символами подстроки и строки. Алгоритм находит позиции, начиная с которых подстрока с учётом до **k** отличий совпадает с фрагментом строки.

Результат работы на примерах из текста к задаче:

input.txt	output.txt
1 0 ababab baaa	1 0
2 1 ababab baaa	2 1 1
3 1 xabcabc ccc	3 0
4 2 xabcabc ccc	4 4 1 2 3 4
5 3 aaa xxx	5 1 0

Вывод по задаче:

Научилась сопоставлять шаблон с учётом k отличий.

Дополнительные задачи

Задача №1 Наивный поиск подстроки в строке

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

- **Формат ввода / входного файла (input.txt).** Первая строка входного файла содержит p , вторая – t . Строки состоят из букв латинского алфавита.
- **Ограничения на входные данные.** $1 \leq |p|, |t| \leq 10^4$.
- **Формат вывода / выходного файла (output.txt).** В первой строке выведите число вхождений строки p в строку t . Во второй строке выведите в возрастающем порядке номера символов строки t , с которых начинаются вхождения p . Символы нумеруются с единицы.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
aba	2
abaCaba	1 5

- Проверяем обязательно – на [OpenEdu](#), курс Алгоритмы программирования и структуры данных, неделя 9, задача 1.

Листинг кода:

```
def main():
    with open('input.txt') as f:
        p = f.readline()[:-1]
        t = f.readline()

    count = 0
    res = ''
    for i in range(len(t) - len(p) + 1):
        if t[i] == p[0]:
            part = t[i:i+len(p)]
            if part == p:
                count += 1
                res += str(i+1) + ' '
    res = res[:-1]

    with open('output.txt', 'w') as f:
        f.write(str(count) + '\n' + str(res))

if __name__ == '__main__':
    main()
```

Пояснение к решению:

Этот код решает задачу поиска подстроки **p** в строке **t** и выводит количество вхождений подстроки, а также индексы (позиции) начала каждого вхождения

Результат работы на примерах из текста к задаче:

input.txt	output.txt
1 aba	1 2
2 abaCaba	2 1 5

Вывод по задаче:

Научилась наивным способом искать подстроки в строке.

Задача №5 Префикс-функция

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

- **Формат ввода / входного файла (input.txt).** Одна строка входного файла содержит s . Строка состоит из букв латинского алфавита.
- **Ограничения на входные данные.** $1 \leq |s| \leq 10^6$.
- **Формат вывода / выходного файла (output.txt).** Выведите значения префикс-функции для всех префиксов строки s длиной $1, 2, \dots, |s|$, в указанном порядке.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt	output.txt	input.txt	output.txt
aaaAAA	0 1 2 0 0 0	abacaba	0 0 1 0 1 2 3

Листинг кода:

```
def main():
    with open('input.txt', 'r') as f_in:
        s = f_in.readline()
        n = len(s)
        pi = [0] * n

    l, r = -1, -1
    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

    with open('output.txt', 'w') as f_out:
        f_out.write(' '.join(list(map(str, pi[:-1]))))

if __name__ == '__main__':
    main()
```

Пояснение к решению:

Этот код реализует алгоритм вычисления префикс-функции строки

Результат работы на примерах из текста к задаче:

≡ input.txt	≡ output.txt
1 aaaAAA	1 0 1 2 0 0 0

≡ input.txt	≡ output.txt
1 abacaba	1 0 0 1 0 1 2 3

Вывод по задаче:

В данной задаче научилась строить префикс-функцию для всех непустых префиксов заданной строки s .

Задача №6 Z-функция

Постройте Z-функцию для заданной строки s .

- **Формат ввода / входного файла (input.txt).** Одна строка входного файла содержит s . Строка состоит из букв латинского алфавита.
- **Ограничения на входные данные.** $2 \leq |s| \leq 10^6$.
- **Формат вывода / выходного файла (output.txt).** Выведите значения Z-функции для всех индексов $1, 2, \dots, |s|$ строки s , в указанном порядке.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt	output.txt	input.txt	output.txt
aaaAAA	2 1 0 0 0	abacaba	0 1 0 3 0 1

- Проверяем **обязательно** – на [OpenEdu](#), курс Алгоритмы программирования и структуры данных, неделя 10,

Листинг кода:

```
def count_z_func(s, n, z):
    ans = []
    l, r = -1, -1
    for i in range(1, n-1):
        if i < r:
            z[i] = min(z[i - l], r - i)
        while i + z[i] < n and s[i + z[i]] == s[z[i]]:
            z[i] += 1
        if i + z[i] - 1 > r:
            l = i
            r = i + z[i] - 1
        ans.append(z[i])

    return " ".join(list(map(str, ans)))

def main():
    with open('input.txt', 'r') as f_in:
        s = f_in.readline()
        n = len(s)
        z = [0] * n

    with open('output.txt', 'w') as f_out:
        res = count_z_func(s, n, z)
        f_out.write(res)

if __name__ == '__main__':
```

```
main()
```

Пояснение к решению:

Этот код реализует вычисление Z-функции строки. Z-функция для строки представляет собой массив, где элемент $z[i]$ равен длине наибольшего префикса строки, который совпадает с суффиксом, начиная с позиции i

Результат работы на примерах из текста к задаче:

<pre>≡ input.txt</pre>	<pre>≡ output.txt</pre>
<pre>1 aaaAAA</pre>	<pre>1 2 1 0 0 0</pre>

<pre>≡ input.txt</pre>	<pre>≡ output.txt</pre>
<pre>1 abacaba</pre>	<pre>1 0 1 0 3 0 1</pre>

Вывод по задаче:

Научилась строить Z-функцию для заданной строки s.

Задача №7 Наибольшая общая подстрока

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.

- **Формат ввода / входного файла (input.txt).** Каждая строка входных данных содержит две строки s и t , состоящие из строчных латинских букв.
- **Ограничения на входные данные.** Суммарная длина всех s , а также суммарная длина всех t не превышает 100 000.
- **Формат вывода / выходного файла (output.txt).** Для каждой пары строк s_i и t_i найдите ее самую длинную общую подстроку и уточните ее параметры, выведя три целых числа: ее начальную позицию в s , ее начальную позицию в t (обе считаются с 0) и ее длину. Формально выведите целые числа $0 \leq i < |s|$, $0 \leq j < |t|$ и $l \geq 0$ такие, что l максимально. (Как обычно, если таких троек с максимальным l много, выведите любую из них.)
- Ограничение по времени. 15 сек.
- Ограничение по памяти. 512 мб.
- Пример:

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

- Объяснение:

Самая длинная общая подстрока первой пары строк – *ool*, она начинается с первой позиции в *toolbox* и с первой позиции в *cool*. Строки из второй строки не имеют общих непустых общих подстрок (в этом случае $l = 0$ и можно вывести любые индексы i и j). Наконец, последние две строки имеют общую подстроку *aab* длины 3, начинающуюся с позиции 0 в первой строке и с позиции 4 во второй. Обратите внимание, что для этой пары строк также можно вывести 2 3 3.

- Что делать?

Для каждой пары строк s и t используйте двоичный поиск, чтобы найти длину наибольшей общей подстроки. Чтобы проверить, есть ли у двух строк общая подстрока длины k ,

- предварительно вычислить хеш-значения всех подстрок длины k из s и t ;
- обязательно используйте несколько хэш-функций (но не одну), чтобы уменьшить вероятность коллизии;
- храните хеш-значения всех подстрок длины k строки s в хеш-таблице; затем пройдите по всем подстрокам длины k строки t и проверьте, присутствует ли хеш-значение этой подстроки в хеш-таблице.

Листинг кода:

```
def main():
    sys.stdin = open('input.txt', 'r')
    with open('output.txt', 'w') as f_out:
        for line in sys.stdin:
            s, t = line.split()
            global n, m
            n, m = len(s), len(t)
            global s1, s2, t1, t2, pow1, pow2
            s1 = get_hashes(s, M1)
            s2 = get_hashes(s, M2)
```



```

t1 = get_hashes(t, M1)
t2 = get_hashes(t, M2)
pow1 = get_pows(max(n, m), M1)
pow2 = get_pows(max(n, m), M2)
l, r = 0, n + 1
while r - l > 1:
    mid = (r + l) // 2
    i1, i2 = check(mid)
    if i1 == -1:
        r = mid
    else:
        l = mid

res = " ".join(map(str, check(l))) + " " + str(l)

f_out.write(res + "\n")

if __name__ == '__main__':
    main()

```

Пояснение к решению:

Этот код выполняет поиск всех подстрок одной строки (s) в другой строке (t) с использованием метода хеширования и бинарного поиска.

Результат работы на примерах из текста к задаче:

input.txt		output.txt	
1	cool toolbox	1	1 1 3
2	aaa bb	2	0 0 0
3	aabaa babbaab	3	2 3 3

Вывод по задаче:

Научилась находить наибольшие общие подстроки.