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 29

Выполнил:

Логачев Д.С.

К3139/К32402

Проверила:

Артамонова В.Е.

Санкт-Петербург

2022 г.

# Содержание отчета

[Содержание отчета 2](#_Toc115810873)

[Задачи по варианту 3](#_Toc115810874)

[4 задача. Равенство подстрок (1.5 балла) 3](#_Toc115810875)

[6 задача. Z-функция (1.5 балла) 4](#_Toc115810876)

[7 задача. Наибольшая общая подстрока (2 балла) 5](#_Toc115810877)

[Вывод 7](#_Toc115810878)

# Задачи по варианту

## 4 задача. Равенство подстрок (1.5 балла)

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.

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 fin:

with open('output.txt', 'w') as fout:

s = fin.readline().strip()

len\_s = len(s)

q = int(fin.readline())

p = 10 \*\* 9 + 7

x = random.randint(1, p - 1)

for i in range(q):

a, b, l = map(int, fin.readline().split())

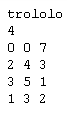
H = PrecomputeHashes(s, len\_s, l, p, x)

if H[a] == H[b]:

fout.write('Yes\n')

else:

fout.write('No\n')



В данном решении основывался на методичных данных из задания.

## 6 задача. Z-функция (1.5 балла)

Постройте Z-функцию для заданной строки s.

def zFunction(word):

z = []

n = len(word)

for i in range(n):

z.append([])

l, r = 0, 0

for i in range(1,n):

if i >= r:

j = 0

while i + j < n and word[i + j] == word[j]:

j += 1

l = i

r = i + j

z[i] = j

else:

if z[i - l] < r - i:

z[i] = z[i - l]

else:

j = r - i

while i + j < n and word[i + j] == word[j]:

j += 1

l = i

r = i + j

z[i] = j

return z

with open("input.txt") as f:

word = f.readline()

z = zFunction(word)

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

f.write(" ".join(list(map(str,z[1:]))))

Просмотрел видеолекцию по z-функции на опенеду и реализовал алгоритм как там.



## 7 задача. 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.

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 fin:

with open('output.txt', 'w') as fout:

while True:

s = fin.readline()

if not s:

exit()

w1, w2 = map(str, s.split())

len\_w1, len\_w2 = len(w1), len(w2)

k = min(len\_w1, len\_w2)

p = 10 \*\* 9 + 7

x = random.randint(1, p - 1)

ok = False

for i in reversed(range(1, k + 1)):

hash\_w1 = PrecomputeHashes(w1, len\_w1, i, p, x)

hash\_w2 = PrecomputeHashes(w2, len\_w2, i, p, x)

for j in range(len(hash\_w1)):

for h in range(len(hash\_w2)):

if hash\_w1[j] == hash\_w2[h]:

fout.write((str(j) + ' ' + str(h) + ' ' + str(i) + '\n'))

ok = True

break

if ok:

break

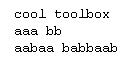
if ok:

break

if not ok:

fout.write((str(0) + ' ' + str(1) + ' ' + str(0) + '\n'))

В данном решении основывались на алгоритме Рабина-Карпа.



# Вывод

В ходе данной лабораторной работы была освоена тема «Подстроки», на практике применены такие методы работы со строками, как алгоритм Рабина-Карпа, Z-функция.