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 27

Выполнил:
Филиппов А.Э.
К3139

Проверила:
Артамонова В.Е.

Санкт-Петербург
2023 г.

Содержание отчета

Задачи по варианту

Задача №2. Карта [1 балл]

В далеком 1744 году во время долгого плавания в руки капитана Александра Смоллетта попала древняя карта с указанием местонахождения сокровищ. Однако расшифровать ее содержание было не так уж и просто. Команда Александра Смоллетта догадалась, что сокровища находятся на x шагов восточнее красного креста, однако определить значение числа она не смогла. По возвращению на материк Александр Смоллетт решил обратиться за помощью в расшифровке послания к знакомому мудрецу. Мудрец поведал, что данное послание таит за собой некоторое число. Для вычисления этого числа необходимо было удалить все пробелы между словами, а потом посчитать количество способов вычеркнуть все буквы кроме трех так, чтобы полученное слово из трех букв одинаково читалось слева направо и справа налево.

Александр Смоллетт догадывался, что число, зашифрованное в послании, и есть число x. Однако, вычислить это число у него не получилось.

После смерти капитана карта была безнадежно утеряна до тех пор, пока не оказалась в ваших руках. Вы уже знаете все секреты, осталось только вычислить число x.

- **Формат ввода / входного файла (input.txt).** В единственной строке входного файла дано послание, написанное на карте.
- **Ограничения на входные данные.** Длина послания не превышает $3 \cdot 105$. Гарантируется, что послание может содержать только строчные буквы английского алфавита и пробелы. Также гарантируется, что послание не пусто. Послание не может начинаться с пробела или заканчиваться им.
- **Формат вывода / выходного файла (output.txt).** Выведите одно число x – число способов вычеркнуть из послания все буквы кроме трех так, чтобы оставшееся слово одинаково читалось слева направо и справа налево.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

• Пример:

input.txt	output.txt	input.txt	output.txt
treasure	8	you will never find the treasure	146

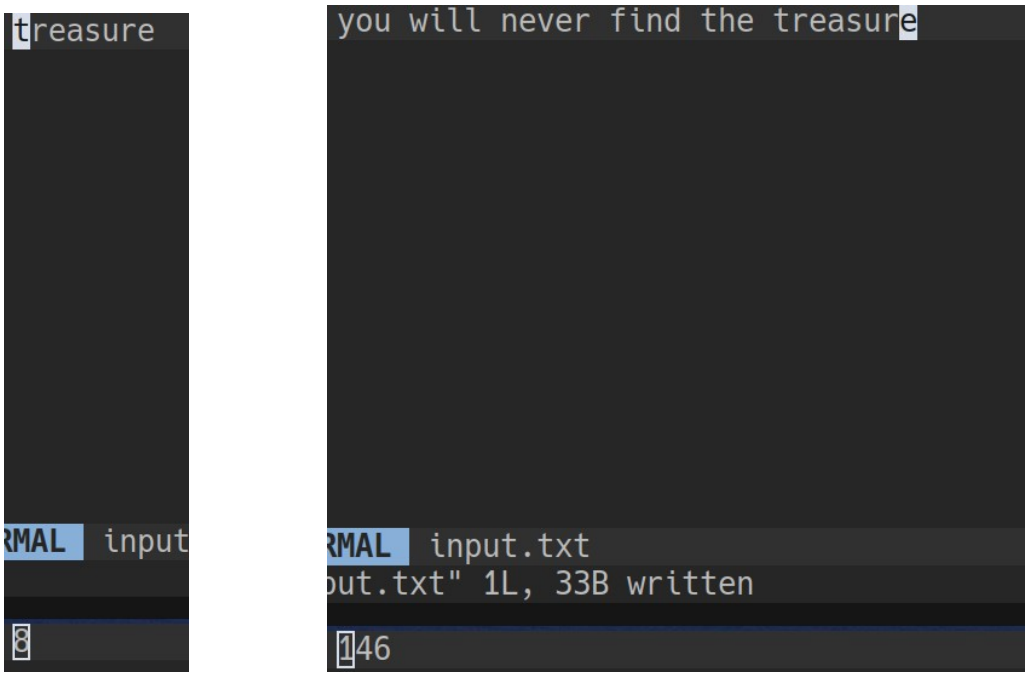
• Проверяем **обязательно** – на OpenEdu, курс Алгоритмы программирования и структуры данных, неделя 9, задача 2.

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

a = lines[0].replace(" ", "")

count = 0

for i in range(len(a) - 2):
    for j in range(i+2, len(a)):
        if a[i] == a[j]:
            count += j - i - 1
out.write(str(count))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

Решение данной задачи реализовано перебором. Сложность алгоритма менее O(n^2), укладывается в временные рамки.
Результат работы кода на примерах из текста задачи:(скрины input output файлов)

Результат работы кода на максимальных и минимальных значениях: (скрины input output файлов)

Проверка задачи на (openedu, acmp и тд при наличии в задаче). (скрин)



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00013426100031210808	13888
Пример из задачи	0.00015295100001821993	13895
Пример из задачи	0.00021753700002591358	13919
Верхняя граница диапазона значений входных данных из текста задачи	1.7262087460003386	15918

Вывод по задаче: Не для всех задач нужна отпимизация. Эта задача спокойно проходит без оптимизации

Задача №6. Z-функция [1.5 балла]

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.** $2 \le |s| \le 106$.

- **Формат вывода / выходного файла (output.txt).** Выведите значения Z-функции для всех индексов 1, 2, ..., $|s|$ строки *s*, в указанном порядке.

  • Ограничение по времени. 2 сек.

  • Ограничение по памяти. 256 мб.

  • Примеры:

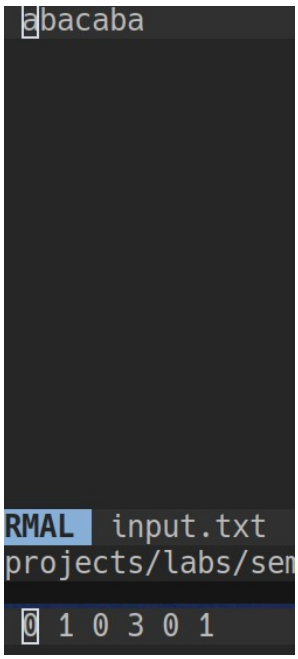
input.txt	output.txt	input.txt	output.txt

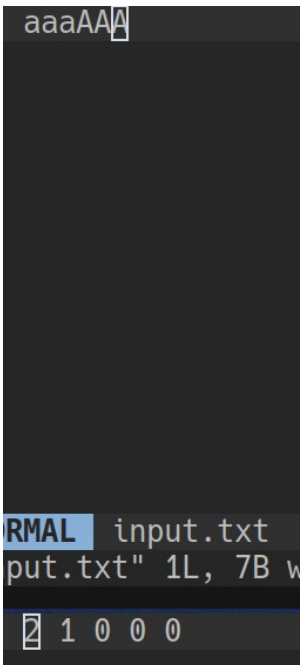
aaaAAA	2 1 0 0 0	abacaba	0 1 0 3 0 1

- Проверяем **обязательно** – на OpenEdu, курс Алгоритмы программирования и структуры данных, неделя 10, задача 2.

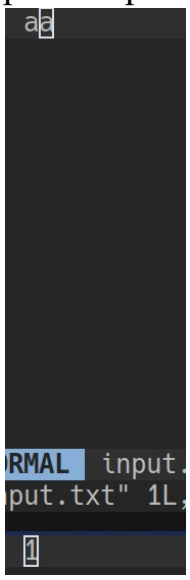
```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

a = lines[0].rstrip()
n = len(a)
z = [0] * n
l = 0
r = 0
for i in range(1, n):
    if i <= r:
        z[i] = min(r-i+1, z[i-l])
    while i+z[i] < n and a[z[i]] == a[i+z[i]]:
        z[i] += 1
    if i + z[i] - 1 > r:
        l = i
        r = i+z[i]-1
out.write(" ".join(map(str, z[1:])))
print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

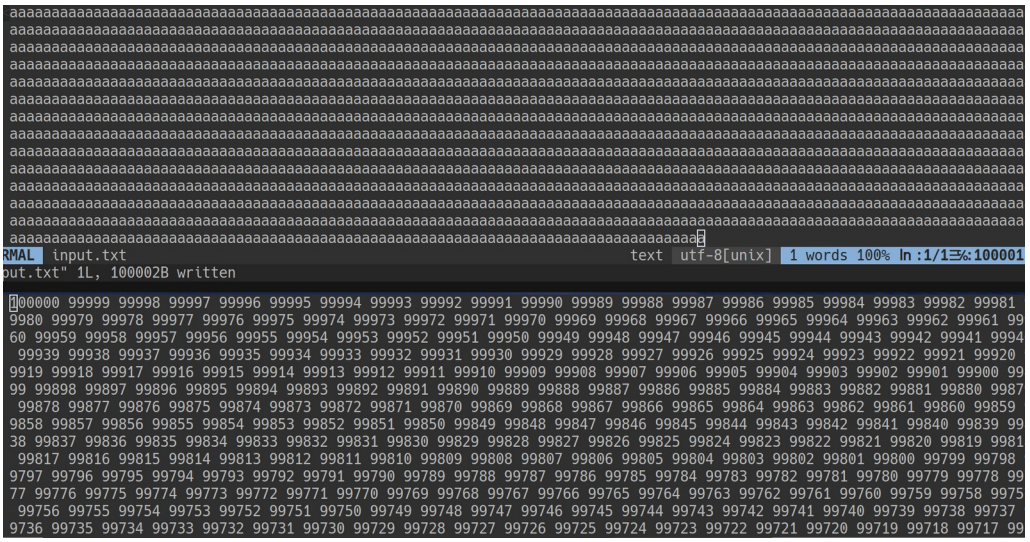
В отличие от наивной реализации Z-функции, данная реализация использует посчитанные до этого значения. Сохраняются левая и правая граница предподсчитанных значений, если мы выходим за пределы границ, то пишем в значение расстояние до границы.
Результат работы кода на примерах из текста задачи:(скрины input output файлов)





Результат работы кода на максимальных и минимальных значениях: (скрины input output файлов)





	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.000164483999469666 74	13889
Пример из задачи	0.000169357001141179 35	13894
Пример из задачи	0.000160613000844023 2	13893
Верхняя граница диапазона значений	0.4377444279998599	10793384

входных данных из текста задачи		

Вывод по задаче: Z-функцию можно оптимизировать используя уже подсчитанные значения функции.

Задача №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.

- **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 *s* не превышает 100 000.
- **Формат вывода / выходного файла (output.txt).** Для каждой пары строк *si* и *ti* 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 $0 \le i < |s|$, $0 \le j < |t|$ и $l \ge 0$ такие, что и 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 aaa bb aabaa babbaab	1 1 3 0 1 0 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 пройдитесь по всем подстрокам длины k строки t и проверьте, присутствует ли хеш-значение этой подстроки в хеш-таблице.

```
import time
import tracemalloc
tracemalloc.start()
```

```
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

# length of the bucket array
CAPACITY = 10000000

# node is a node of linked list (I just don't wanna create LinkedList
class)


class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None

    def __repr__(self):
        return str(self.value)

    def __str__(self):
        return f"key: {self.key}, value: {self.value}, next: {self.next}"


class HashTable:
    def __init__(self):
        self.size = 0
        self.buckets = [None]*CAPACITY

    def hash(self, key):
        summ = 0
        for i in range(len(key)):
            summ += (ord(key[i])) * (125 ** i)

        return summ % CAPACITY

    def insert(self, key, value):
        self.size += 1
        index = self.hash(key)
        node = self.buckets[index]

        if node is None:
            self.buckets[index] = Node(key, value)
        else:
            prev = node
            while node is not None:
                prev = Node
                node = node.next
            prev.next = Node(key, value)

    def get(self, key):
```

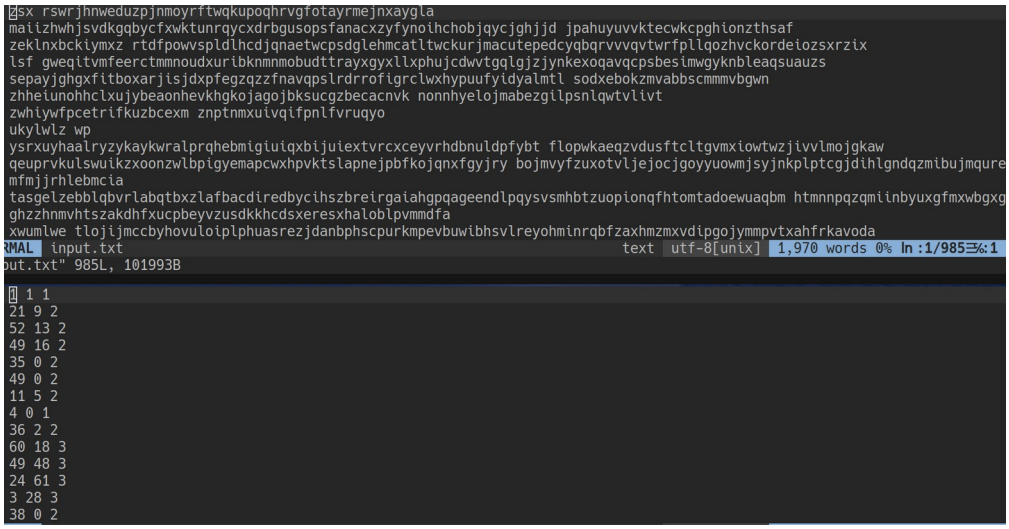
```
        index = self.hash(key)
        node = self.buckets[index]

        while node is not None and node.key != key:
            node = node.next

        return node.value if node is not None else None

    def exists(self, key):
        return True if self.get(key) != None else False

    def remove(self, key):
        index = self.hash(key)
        node = self.buckets[index]

        # persist previous element, so we can remove linked list node
        prev = None

        while node is not None and node.key != key:
            prev = node
            node = node.next
        if node is not None:
            self.size -= 1
            if prev is None:
                self.buckets[index] = node.next
            else:
                prev.next = node.next


table = HashTable()


def existsSubstringOfLengthK(s, t, k):
    # insert all k substrings to our hashtable
    for i in range(len(s) - k + 1):
        table.insert(s[i:k+i], i)
    for i in range(len(t) - k + 1):
        string = t[i:k+i]
        string_s = table.get(string)
        if string_s != None:
            return (string_s, i)
    return None


for i in range(len(lines)):
    s, t = lines[i].split()
    k = min(len(s), len(t))

    l = 0
    r = k
    m = 0
    mem = None
    while l <= r:
```

```
        m = (l + r) // 2

        res = existsSubstringOfLengthK(s, t, m)
        if res != None:
            mem = res
            l = m + 1

        else:
            r = m - 1
    out.write(str(mem[0]) + " " + str(mem[1]) + " " + str(r) + "\n")


print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

Используя двоичный поиск перебираем максимально возможную длину подстроки. Для того, чтобы проверить есть ли подстрока в строке s и строке t записываем все подстроки строки s в хэш-таблицу, далее проходим по строке t и проверяем встречалась ли эта подстрока в строке s через хэш-таблицу.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

Результат работы кода на максимальных и минимальных значениях: (скрины input output файлов)



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0139888899999382081	80020136
Пример из задачи	0.024863461998393177	80023484
Верхняя граница диапазона значений входных данных из текста задачи	4.553469309999855	111798777

Вывод по задаче:
Для решения этой задачи пришлось применить много приемов, изученных в первом семестре. Например это хэш-таблица и двоичный поиск

Вывод

Вот и подошел к концу предмет «Алгоритмы и структуры данных». Последняя лабораторная работа отточила мои навыки использования алгоритмов и структур данных, были использованы структуры и алгоритмы, входящие в курс первого семестра. Но были и идеи схожие с динамическим программированием второго семестра. В целом — отличная практика для того, чтобы освежить забытое старое и закрепить изученный материал.