

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

| | |
|---|-------------|
| Содержание отчета | 2 |
| Задачи по варианту | 3-14 |
| Задача №2. Карта [1 балл] | 3-6 |
| Задача №6. Z-функция [1.5 балла] | 6-9 |
| Задача №7. Наибольшая общая подстрока [2 балла] | 9-14 |
| Вывод | 15 |

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 $3 \cdot 10^5$. 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

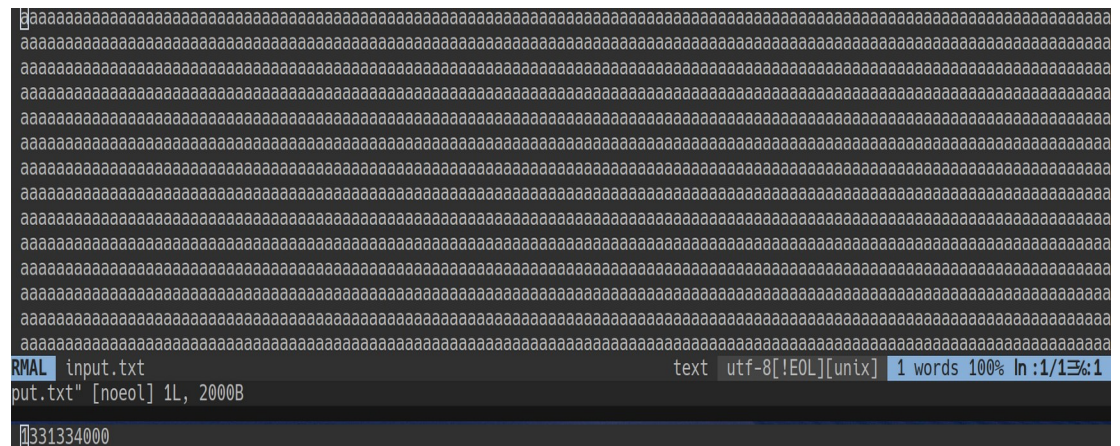
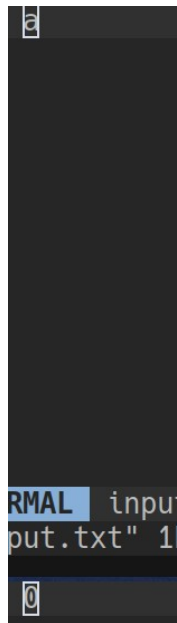
for i in range(len(a) - 2):
    for j in range(i+2, len(a)):
        if a[i] == a[j]:
            count += j - i - 1
out.write(str(count))

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

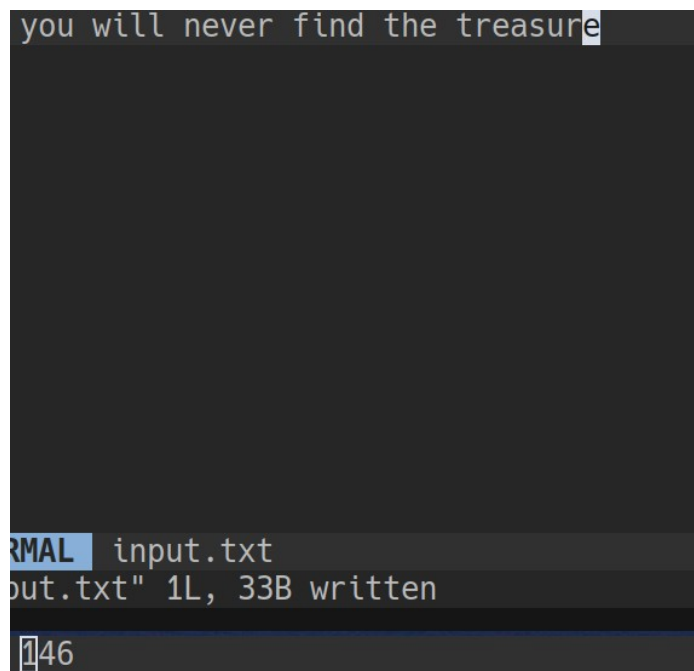
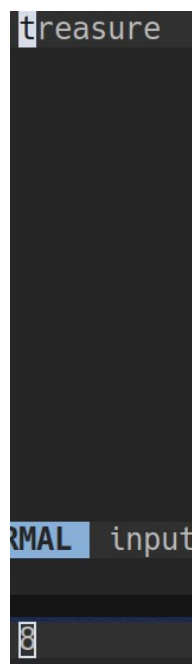
Решение данной задачи реализовано перебором. Сложность алгоритма менее $O(n^2)$, укладывается в временные рамки.

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, астр 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 оптимизация. 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.** $2 \leq |s| \leq 10^6$.
- **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.t | output.t | input.t | output.tx |
| xt | xt | xt | t |

| | | | |
|-------|---------|--------|-----------|
| aaaAA | 2 1 0 0 | abacab | 0 1 0 3 0 |
| A | 0 | a | 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)

```
abacaba
RMAL input.txt
projects/labs/sen
1 0 3 0 1
```

```
aaaAAa
RMAL input.txt
put.txt" 1L, 7B w
2 1 0 0 0
```

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

```
aa
RMAL input.txt
put.txt" 1L, 100002B written
text utf-8[unix] 1 words 100% ln:1/13%100001
00000 99999 99998 99997 99996 99995 99994 99993 99992 99991 99990 99989 99988 99987 99986 99985 99984 99983 99982 99981
99980 99979 99978 99977 99976 99975 99974 99973 99972 99971 99970 99969 99968 99967 99966 99965 99964 99963 99962 99961 99
60 99959 99958 99957 99956 99955 99954 99953 99952 99951 99950 99949 99948 99947 99946 99945 99944 99943 99942 99941 99940
99939 99938 99937 99936 99935 99934 99933 99932 99931 99930 99929 99928 99927 99926 99925 99924 99923 99922 99921 99920
99919 99918 99917 99916 99915 99914 99913 99912 99911 99910 99909 99908 99907 99906 99905 99904 99903 99902 99901 99900 99
99 99898 99897 99896 99895 99894 99893 99892 99891 99890 99889 99888 99887 99886 99885 99884 99883 99882 99881 99880 99879
99878 99877 99876 99875 99874 99873 99872 99871 99870 99869 99868 99867 99866 99865 99864 99863 99862 99861 99860 99859
99858 99857 99856 99855 99854 99853 99852 99851 99850 99849 99848 99847 99846 99845 99844 99843 99842 99841 99840 99839 99
38 99837 99836 99835 99834 99833 99832 99831 99830 99829 99828 99827 99826 99825 99824 99823 99822 99821 99820 99819 99818
99817 99816 99815 99814 99813 99812 99811 99810 99809 99808 99807 99806 99805 99804 99803 99802 99801 99800 99799 99798
99797 99796 99795 99794 99793 99792 99791 99790 99789 99788 99787 99786 99785 99784 99783 99782 99781 99780 99779 99778 99
77 99776 99775 99774 99773 99772 99771 99770 99769 99768 99767 99766 99765 99764 99763 99762 99761 99760 99759 99758 99757
99756 99755 99754 99753 99752 99751 99750 99749 99748 99747 99746 99745 99744 99743 99742 99741 99740 99739 99738 99737
99736 99735 99734 99733 99732 99731 99730 99729 99728 99727 99726 99725 99724 99723 99722 99721 99720 99719 99718 99717 99
```

| | Время выполнения (с) | Затраты памяти (байт) |
|---|------------------------|-----------------------|
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00016448399946966674 | 13889 |
| Пример из задачи | 0.00016935700114117935 | 13894 |
| Пример из задачи | 0.0001606130008440232 | 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 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 | 1 1 3 |
| toolbox | 0 1 |
| aaa bb | 0 |
| aabaa | 0 4 |
| babbaab | 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.

```
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)

The screenshot shows a terminal window with the following content:

```

ool toolbox
aaa bb
aabaa babbaab

```

Below the input files, there is a section labeled "RMAL" with the filename "input.txt".

At the bottom, there is a table of results:

| | | |
|---|---|---|
| 1 | 1 | 3 |
| 0 | 0 | 0 |
| 2 | 3 | 3 |

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

```

[5] rswrjhnwedu3pjmoyrftwqkupoqhrvgrotayrmejnxaygla
mailzhwhjsvdkgbycfxwktunrkyxdrbgusopsfanacxyfynoihchobjgycjghjjd jpahuyuvvktecwkcpgghionzthsaf
zeklnxbcklymxz rtdfpowvpldlhcdjgnaetwcpdglehmcaltwckurjmacutepedcyqbrvvvqvtwrfllqozhvckordeiozsrzix
lsf gweqitvmfeerctmnmoudxuribknmmobudttrayxgyllxphujcdwtgqlgjzjynkexogavqcpbesimwyknbleaqsauzs
sepayjghgxfitboxarjtsjdxpfezqzzfnavqpslrdrrrofigrclwxhyuufyidyaltl sodxebokzmvabbscmmmvbgwn
zhhelunohhclxujybeaonhevkhgkojagojbksucgzbecacnvk nonnyelojmabezgilpsnlqwtvlvt
zwhlywfpctrifkuzbcxm znptnmxulvqlfplnfvrugyo
ukylwlz wp
ysrxuyhaalryzykaykwalprqhebmigiuiqxbijulextvrcxeyvrhdbnuldpybt flopwkaeqzvdusftcltgvmxiovtwzjivvlmojgkaw
qeuoprkvulsuikzxoonzwlbpigyemapcxhpkvtslapnejpbfojqnxfgyjry bojmvvyfzuxotvljejocjgoyuowmjsyjnklptcgjdthlgndqzmibujmqure
mfmjrrhlebmcia
tasgelzebbqlbvrlabqtbxzlafbacdiaredbycihszbreirgaiahgppageendlpqysvsmhbtzuopionqfhtomtadoewuaqbm htmnnpqzqmibnyuxgfmxbwgxg
ghzzhnmvhtszakdhfxucpbeyvzusdkkcdsxeresxhaloblpvmmdfa
xwumlwe tloijmccbyhovuloiplhuasrezjdanbphscpurkmpvbwibhsvlreyohminrqbfzaxhmzmxdipgojymmpvtxahfrkavoda
RMAL input.txt text utf-8[unix] 1,970 words 0% ln:1/985
out.txt" 985L, 101993B
1 1
21 9 2
52 13 2
49 16 2
35 0 2
49 0 2
11 5 2
4 0 1
36 2 2
60 18 3
49 48 3
24 61 3
3 28 3
38 0 2

```

| | Время выполнения (с) | Затраты памяти (байт) |
|--|----------------------|-----------------------|
| Нижняя граница диапазона значений входных данных из текста задачи | 0.013988889999382081 | 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.