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

Отчет по лабораторной работе №8
по курсу «Алгоритмы и структуры данных»
Тема: Динамическое программирование №1
Вариант 27

Выполнил:
Филиппов А.Э.
К3139

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Содержание отчета **2**

Задачи по варианту **3-9**

Задача №3. Редакционное расстояние 3-6

Задача №4. Наибольшая общая подпоследовательность двух... 6-9

Вывод **10**

Задачи по варианту

Задача №3. Редакционное расстояние

Редакционное расстояние между двумя строками – это минимальное количе ство операций (вставки, удаления и замены символов) для преобразования одной строки в другую. Это мера сходства двух строк. У редакционного расстояния есть применения, например, в вычислительной биологии, обработке текстов на есте ственном языке и проверке орфографии. Ваша цель в этой задаче – вычислить расстояние редактирования между двумя строками.

- **Формат ввода / входного файла (input.txt).** Каждая из двух строк ввода содержит строку, состоящую из строчных латинских букв. Длина обеих строк - от 1 до 5000.
- **Формат вывода / выходного файла (output.txt).** Выведите расстояние ре дактирования между заданными двумя строками.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt	output.txt	input.txt	output.txt	input.txt	output.txt
ab ab	0	short ports	3	editing distance	5

⋆ Редакционное расстояние во втором примере равна 3:

s	h	o	r	t	-
-	p	o	r	t	s

- Дополнительное задание. Выведите в ответ также построчно, какие действия необходимо сделать, чтобы из 1 строки получилась вторая, порядок записи действий - произвольный. Например:

input.txt	output.txt
short ports	3 del s change h p add s

Как вариант, можно вместо действий вывести табличку как в пункте со звездочкой ⋆.

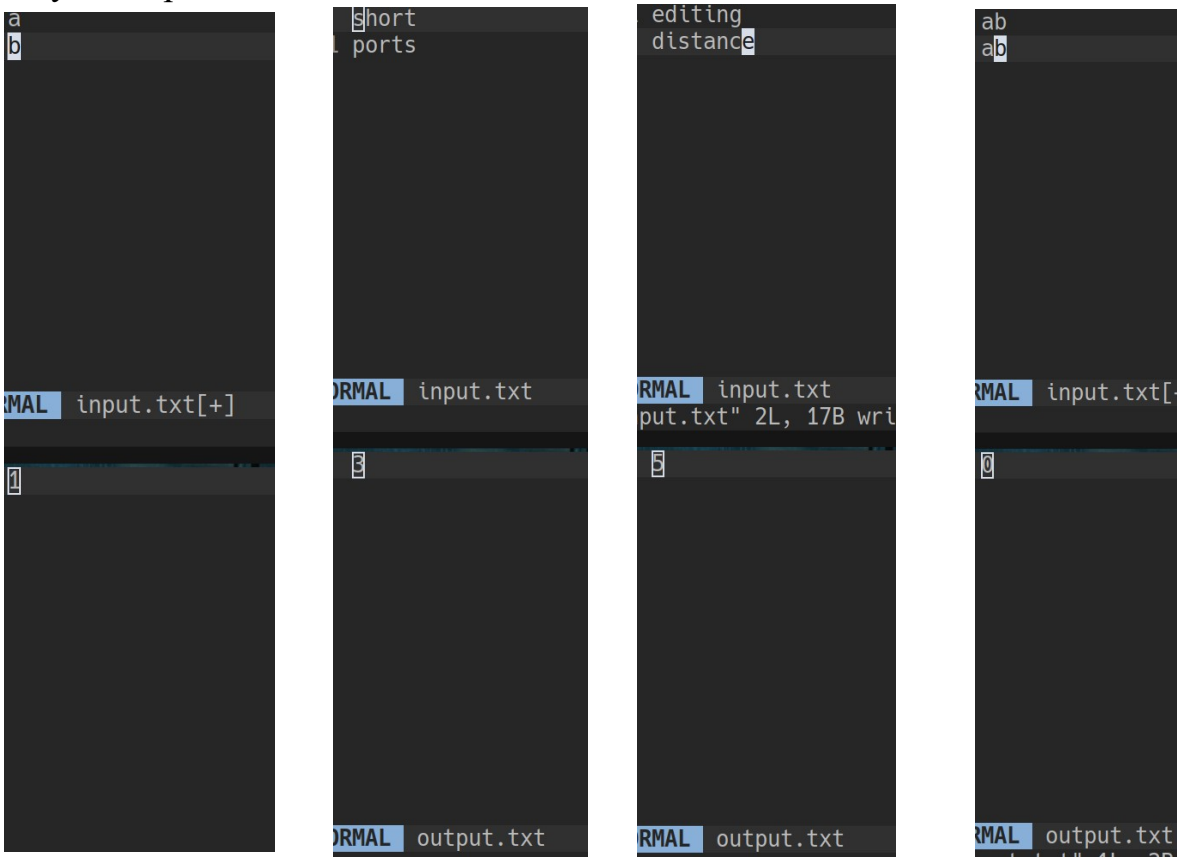
Код:

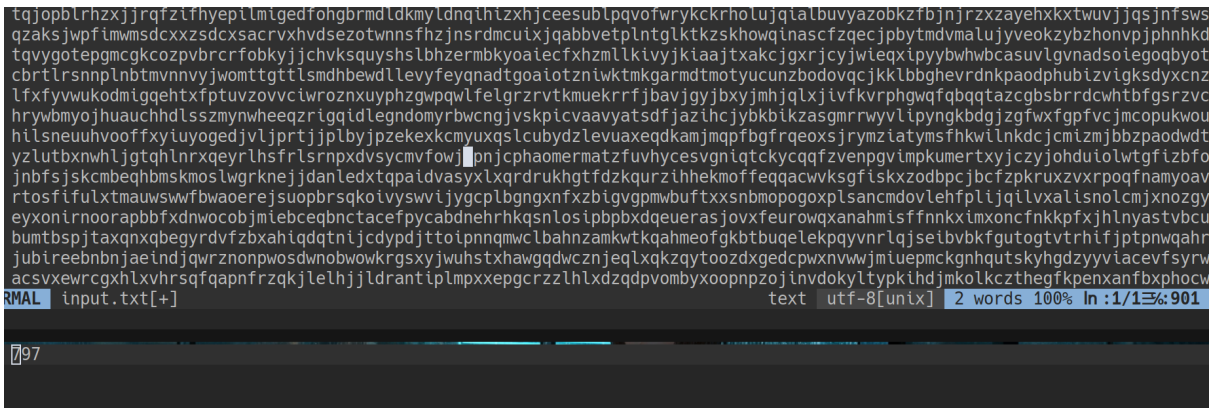
```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

a = lines[0]
b = lines[1]

n, m = len(a), len(b)
if n > m:
    a, b = b, a
    n, m = m, n

curr_row = []
for i in range(n+1):
    curr_row.append(i)

for i in range(1, m + 1):
    prev_row, curr_row = curr_row, [i] + [0] * n
    for j in range(1, n + 1):
        move_aus_a = prev_row[j] + 1
        move_aus_b = curr_row[j-1] + 1
        fortsetzen = prev_row[j-1]
        if a[j-1] != b[i-1]:
            fortsetzen += 1
        curr_row[j] = min(move_aus_a, move_aus_b, fortsetzen)

out.write(str(curr_row[n]))
```

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

Для решения данной задачи использовалось расстояние Левенштайна. Вместо того, чтобы сохранять массив для ускорения работы сохраняются последняя строка и столбец.

Результат работы кода:





	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00017152299915323965	13939
Пример из задачи	0.0001822340000217082	13941
Пример из задачи	0.0001964399998541921 4	13947
Пример из задачи	0.0002520400003049872	13952
Верхняя граница диапазона значений входных данных из текста задачи	1.9091916950001178	92904

Вывод по задаче: Расстояние Левенштайна считается самым эффективным алгоритмом для расчета редакционного расстояния. Его сложность - O(n^2)

Задача №4. Наибольшая общая подпоследовательность двух по следовательностей

Вычислить длину самой длинной общей подпоследовательности из двух по следовательностей.

Даны две последовательности $A = (a1, a2, ..., an)$ и $B = (b1, b2, ..., bm)$, найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотри цательное целое число p такое, что существуют индексы $1 \le i1 < i2 < ... < ip \le n$ и $1 \le j1 < j2 < ... < jp \le m$ такие, что $ai1 = bj1, ..., aip = bjp$.

- **Формат ввода / входного файла (input.txt).**
  - Первая строка: n - длина первой последовательности.
  - Вторая строка: $a1, a2, ..., an$ через пробел.
  - Третья строка: m - длина второй последовательности.

– Четвертая строка: $b1, b2, ..., bm$ через пробел.

• Ограничения: $1 \le n, m \le 100$; $-109 < ai, bi < 109$.

• **Формат вывода / выходного файла (output.txt).** Выведите число

p. • Ограничение по времени. 1 сек.

• Примеры:

input.txt	output.txt	input.txt	output.txt	input.txt	output.txt
3 2 7 5 2 2 5	2	1 7 4 1 2 3 4	0	4 2 7 8 3 4 5 2 8 7	2

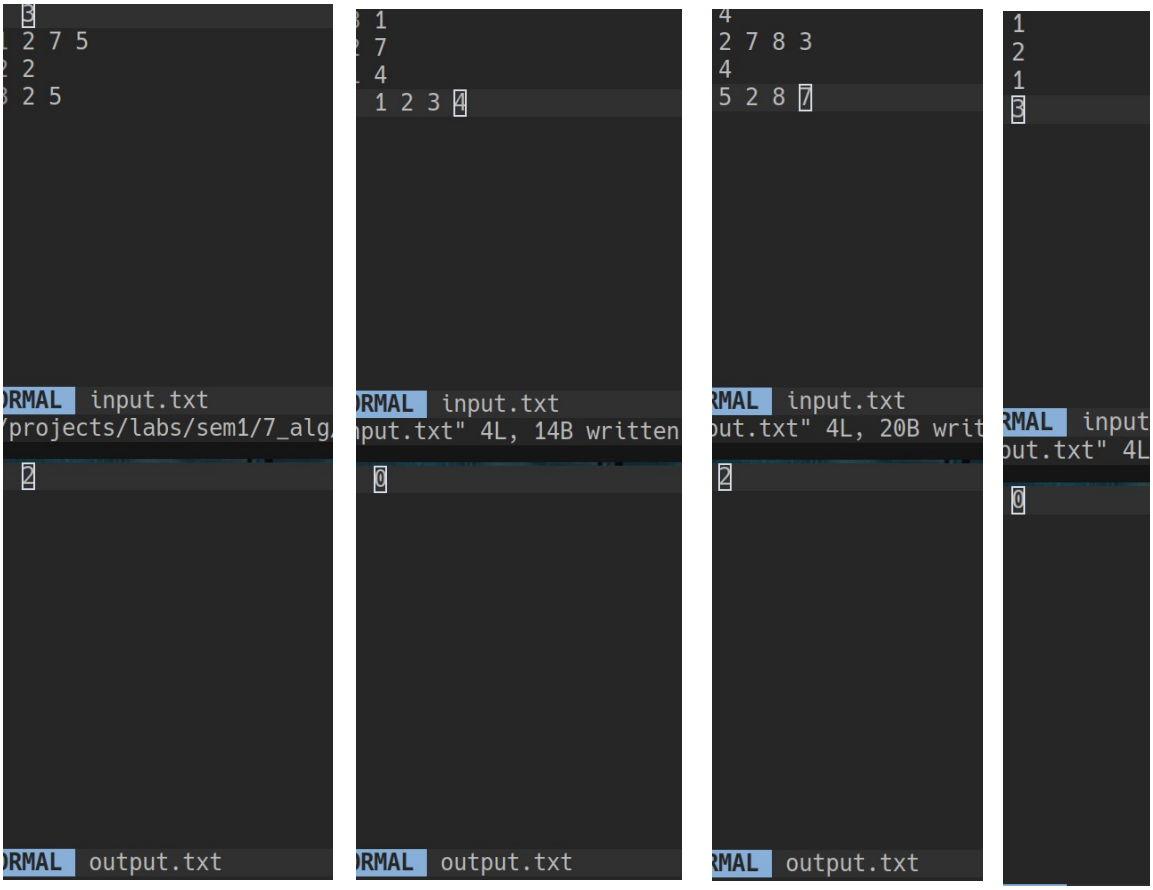
• В первом примере одна общая подпоследовательность – (2, 5) длиной 2, во втором примере две последовательности не имеют одинаковых элементов. В третьем примере - длина 2, последовательности – (2, 7) или (2, 8).

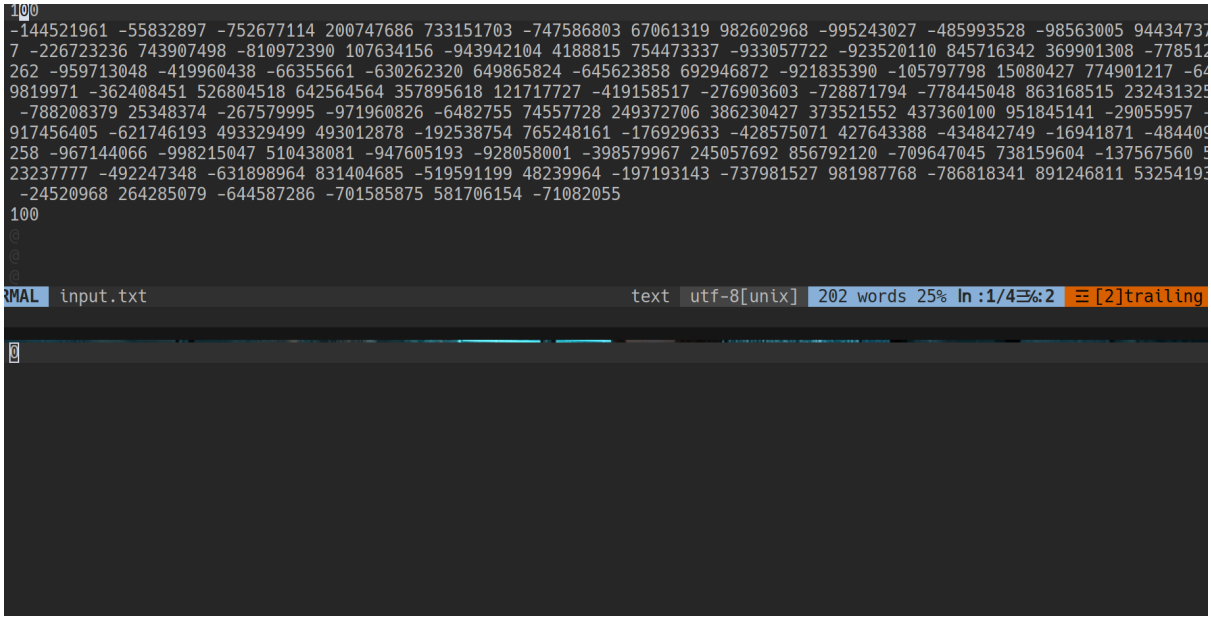
Код:

```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

n = int(lines[0])
a = list(map(int, lines[1].split()))

m = int(lines[2])
b = list(map(int, lines[3].split()))

dp = [[None for x in range(m+1)] for i in range(n+1)]

for i in range(n+1):
    for j in range(m+1):
```

```
        if i == 0 or j == 0:
            dp[i][j] = 0
        elif a[i-1] == b[j-1]:
            dp[i][j] = dp[i-1][j-1] + 1
        else:
            dp[i][j] = max(dp[i-1][j], dp[i][j-1])

out.write(str(dp[n][m]))

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

Для решения данной задачи использовалась динамическая матрица. В случае если строки совпадают, то значение увеличивается на единицу, иначе берется максимум из соседней левой или соседней верхней клетки.
Результат работы кода:





	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.000200149999727727 85	14041
Пример из задачи	0.000207376000616932 3	14047
Пример из задачи	0.000191766999705578 2	14047
Пример из задачи	0.000212162000025273 3	14053
Верхняя граница диапазона значений входных данных из текста задачи	0.009543416999804322	115035

Вывод по задаче: К некоторым задачам можно подходить с помощью построения динамической матрицы вместо рекурсии. Решение можно далее оптимзировать сохраняя лишь последнюю строку и столбец.

Вывод

Подход динамического программирования чем-то похож на модель «разделяй и властвуй». На первый взгляд сложно выделить какие-то отличия между этими двумя подходами. Отличие же заключается в том, что «разделяй и властвуй» разбивает большую задачу на маленькие «сверху», а затем объединяет их в одно решение. В динамическом программировании, как правило большая задача решается «снизу», поэтому подзадачи объединяются естественнее и проще.