

Национальный исследовательский Университет ИТМО  
Мегафакультет информационных и трансляционных технологий  
Факультет инфокоммуникационных технологий

# Алгоритмы и структуры данных

Лабораторная работа №1

**Работу**

**выполнил:**

А.Э. Филиппов

Группа: К3139

**Преподаватель:**

Д. Добриборщ

Санкт-Петербург  
2022

# Содержание

|  |          |
|--|----------|
| <b>1. Задачи по варианту</b>                                     | <b>3</b> |
| 1.1. Задача №3. Редакционное расстояние . . . . .                | 3        |
| 1.2. Задача №4. Наибольшая общая подпоследовательность . . . . . | 5        |
| <b>Заключение</b>  | <b>9</b> |

# 1. Задачи по варианту

## 1.1. Задача №3. Редакционное расстояние

Редакционное расстояние между двумя строками – это минимальное количество 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 естественном 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 редактирования 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        | 0          | shorts    | 3          | editing   | 5          |
| ab        |            | ports     |            | distance  |            |

- Редакционное расстояние во втором примере равно 3:

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

Код:

```
1  import time
2  import tracemalloc
3
4  tracemalloc.start()
5  t_start = time.perf_counter()
6  file = open('input.txt')
7  lines = file.readlines()
8  out = open("output.txt", "w")
9
10 a = lines[0]
11 b = lines[1]
12
13 n, m = len(a), len(b)
14 if n > m:
15     a, b = b, a
16     n, m = m, n
17
18 curr_row = []
19 for i in range(n+1):
20     curr_row.append(i)
21
22 for i in range(1, m + 1):
```

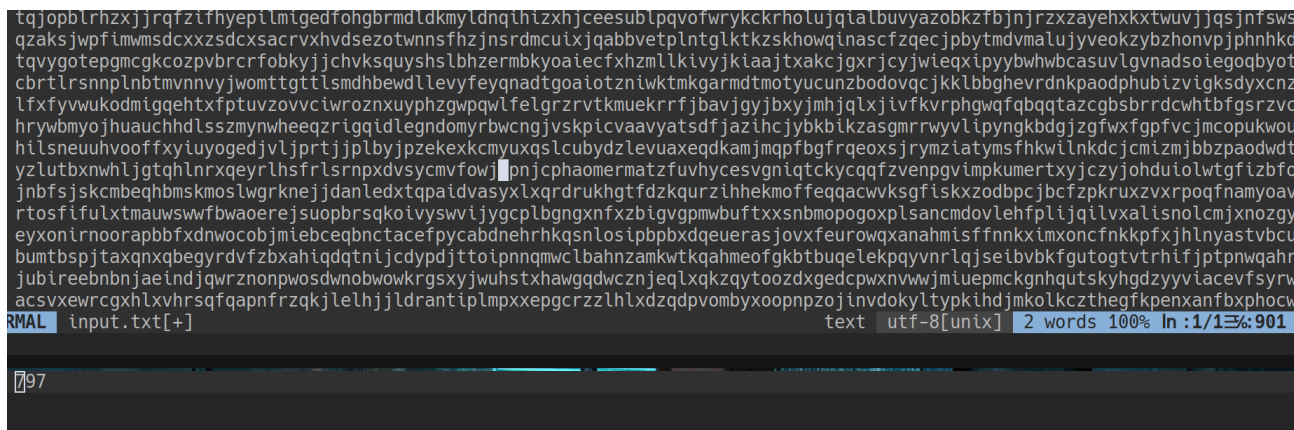
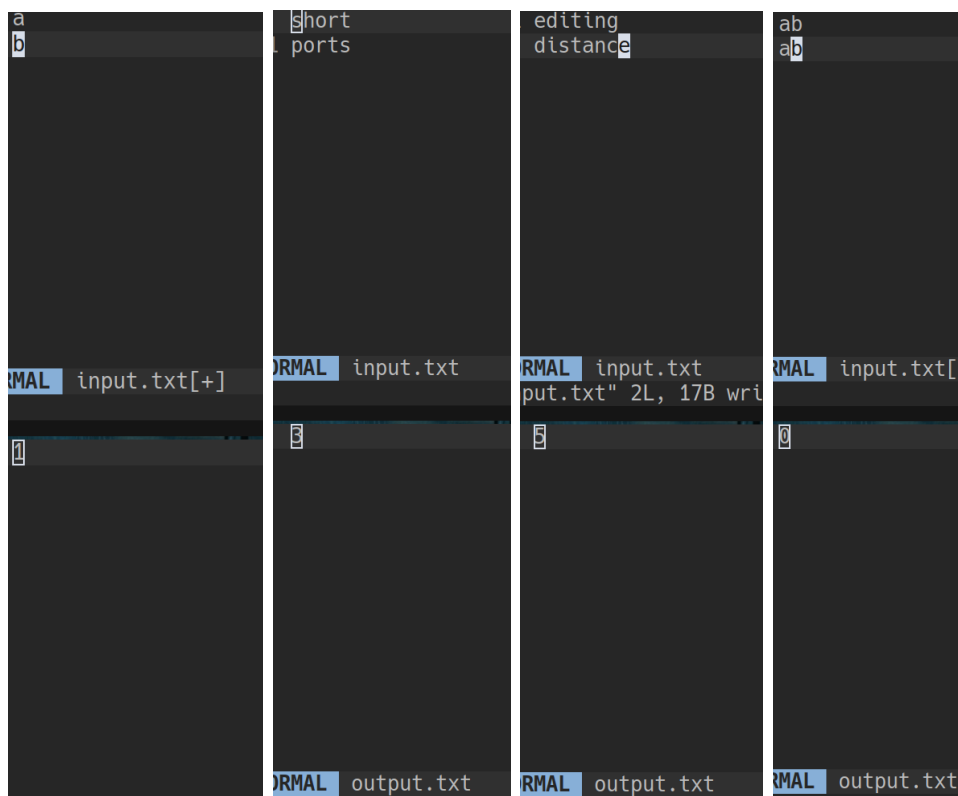
```

23     prev_row, curr_row = curr_row, [i] + [0] * n
24     for j in range(1, n + 1):
25         move_aus_a = prev_row[j] + 1
26         move_aus_b = curr_row[j-1] + 1
27         fortsetzen = prev_row[j-1]
28         if a[j-1] != b[i-1]:
29             fortsetzen += 1
30         curr_row[j] = min(move_aus_a, move_aus_b, fortsetzen)
31
32 out.write(str(curr_row[n]))
33

```

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 1   | 0.0001822340000217082  | 13941                 |
| Пример из задачи 2   | 0.00019643999985419214 | 13947                 |
| Пример из задачи 3   | 0.0002520400003049872  | 13952                 |
| Верхняя граница диапазона значений входных данных из текста задачи | 1.9091916950001178     | 92904                 |

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

## 1.2. Задача №4. Наибольшая общая подпоследовательность

Вычислить длину самой длинной общей подпоследовательности из двух последовательностей. Даны две последовательности  $A = (a_1, a_2, \dots, a_n)$  и  $B = (b_1, b_2, \dots, b_m)$ , найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицательное целое число  $p$  такое, что существуют индексы  $1 \leq i_1 < i_2 < \dots < i_p \leq n$  и  $1 \leq j_1 < j_2 < \dots < j_p \leq m$  такие, что  $a_{i_1} = b_{j_1}, \dots, a_{i_p} = b_{j_p}$ .

- **Формат входного/выходного файла (input.txt)**

- Первая строка  $n$  - длина первой последовательности
- Вторая строка  $a_1, a_2, \dots, a_n$  через пробел
- Третья строка  $m$  - длина второй последовательности
- Первая строка  $b_1, b_2, \dots, b_m$  через пробел

- Ограничения:  $1 \leq n, m \leq 100$ ;  $-109 < a_i, b_i < 109$

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

- Ограничение по времени. 1 сек.

- Примеры

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

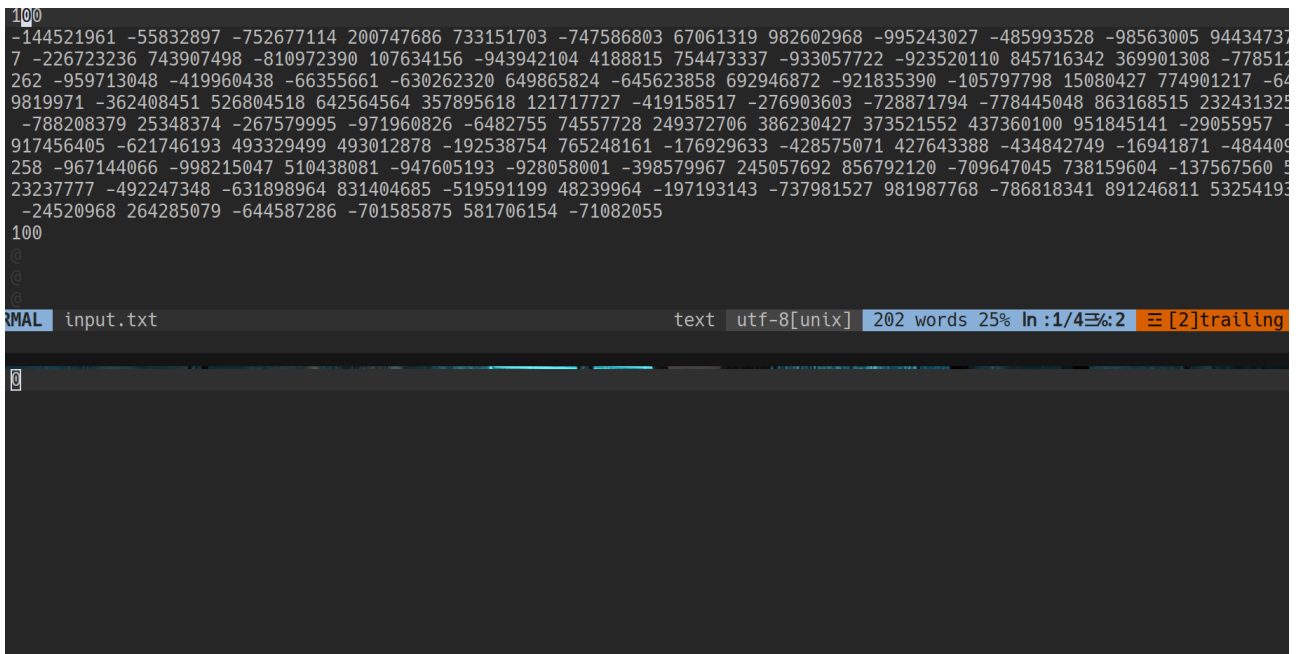
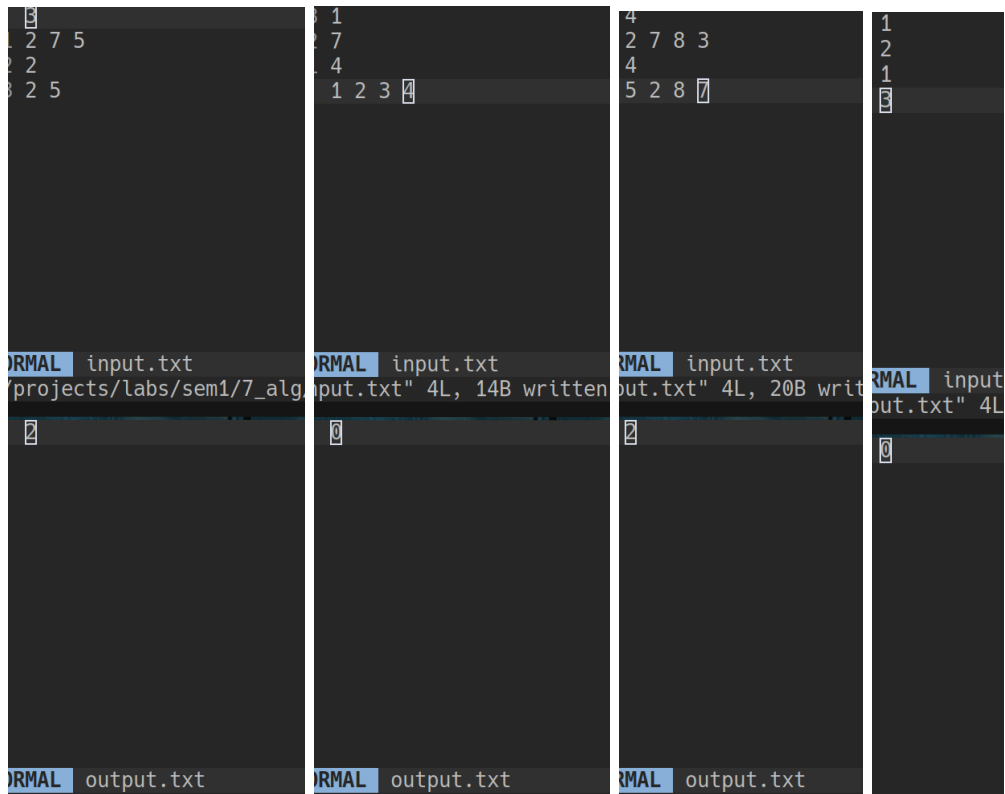
- В первом примере одна общая подпоследовательность – (2, 5) длиной 2, во втором примере две последовательности не имеют одинаковых элементов. В третьем примере - длина 2, последовательности – (2, 7) или (2, 8).

## Код:

```
1 import time
2 import tracemalloc
3
4 tracemalloc.start()
5 t_start = time.perf_counter()
6 file = open('input.txt')
7 lines = file.readlines()
8 out = open("output.txt", "w")
9
10 n = int(lines[0])
11 a = list(map(int, lines[1].split()))
12
13 m = int(lines[2])
14 b = list(map(int, lines[3].split()))
15
16 dp = [[None for x in range(m+1)] for i in range(n+1)]
17
18 for i in range(n+1):
19     for j in range(m+1):
20         if i == 0 or j == 0:
21             dp[i][j] = 0
22         elif a[i-1] == b[j-1]:
23             dp[i][j] = dp[i-1][j-1] + 1
24         else:
25             dp[i][j] = max(dp[i-1][j], dp[i][j-1])
26
27 out.write(str(dp[n][m]))
28
29 print("Время выполнения: " + str(time.perf_counter() - t_start) +
30       ↵ " секунд")
31 print("Использование памяти: " +
32       str(tracemalloc.get_traced_memory()[1]) + " байт")
33 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  | 0.00020014999972772785 | 14041                 |
| Пример из задачи 1   | 0.0002073760006169323  | 14047                 |
| Пример из задачи 2   | 0.0001917669997055782  | 14047                 |
| Пример из задачи 3   | 0.0002121620000252733  | 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 оптимизировать 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.