

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 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 ab	0	short ports	3	editing distance	5

- ★ Редакционное расстояние во втором примере равно 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)

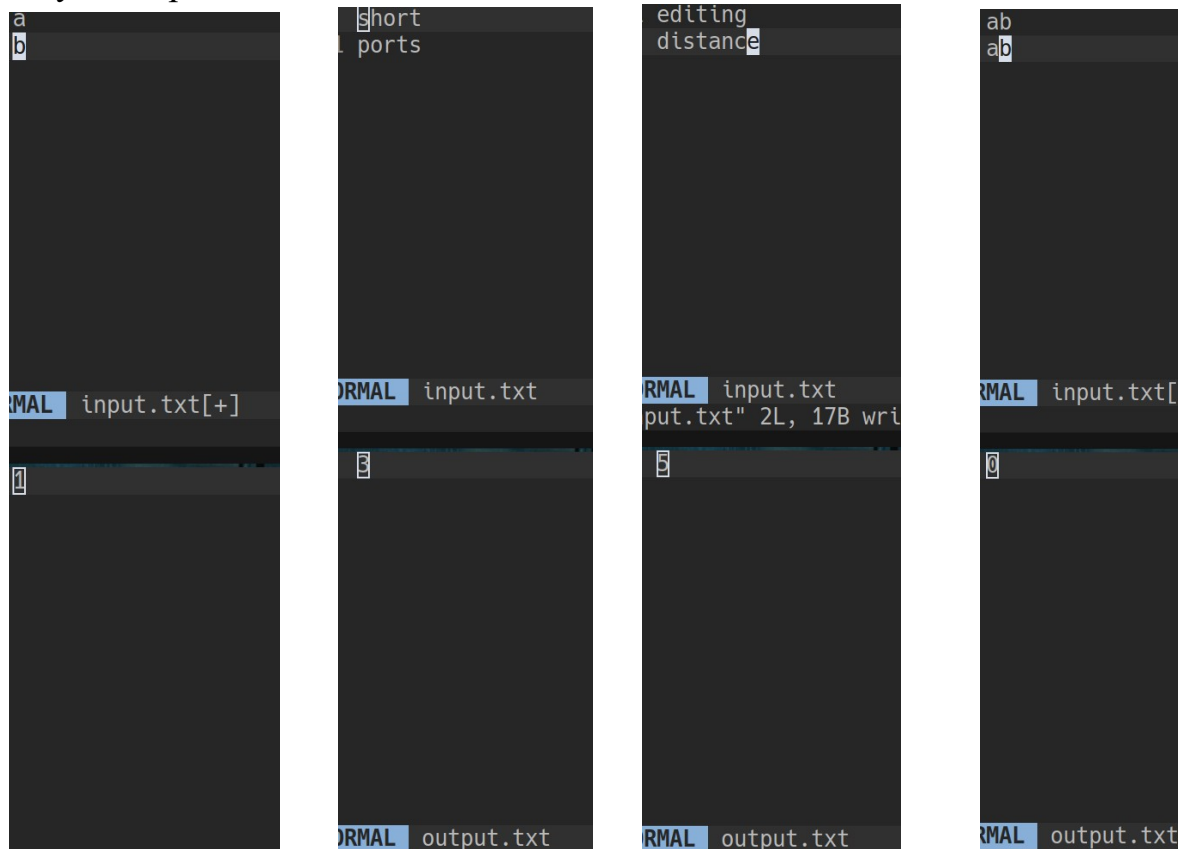
for i in range(1, m + 1):
    prev_row, curr_row = curr_row, [i] + [0] * n
    for j in range(1, n + 1):
        move_aus_a = prev_row[j] + 1
        move_aus_b = curr_row[j-1] + 1
        fortsetzen = prev_row[j-1]
        if a[j-1] != b[i-1]:
            fortsetzen += 1
        curr_row[j] = min(move_aus_a,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



```
tqjopblrhxjrrqfzlfhyeplmtlgedfongbrmdldkmyldnqlhtzxhjceesublpqovfwrykckrholujqialbuvyazobkzfbjnrxzayehxktwuvjjqsjnfsws
qzaksjwpfllmwsdcxxzsdxcxsacrvxhvdsezotwnnsfhzjnsrdmcuixjqabbvetplntglktkzskhowqinascfzqecjpytmdvmlujyveokzyzbzhonvpjphnhkd
tvyvgotepgmcgkcozpvbrcrfobkyjjchvksquyshslbhzermbykyaiecfzhxmllkivyjkliaajtxakcjgxrjcyjwieqxiipybwhwbcasuvlgvnadsoieqogbyot
cbrtlrsnplnbtmnnvyjwomttgttlsmdbewdllevyfeynadtgoaitznltkmgarmdmtotucunzboovqckllbbghevrdnkpaodphubizvlgksdyxcnz
lfxfyvwukodmiggehtxftptuvzovvciwroznxyphzgwpqwlflgrzrvtkmuekrfjbavjgyjbyxymhjqlxjivfkvrphgwfqbqqtazcgsbrddcwhtbfgsrzvc
hrywbmyojhuauchhdlsszmynwheeqzrigqldlegndomyrbwcnvjvskpicvaavyatsdfjazihcjybkikzasgmrrwyvlypyngkbgdjzgfxfpvcjmcopukwou
hilsneuuhvooffxyiuyogedjvljprtjjplbyjpzekexkcmuxqslcubydzlevuaxeqdkamjmqpfbgfrqeoxxjrymziatymshfkwlknkdcjcmizmjbzpaodwdt
yzlbtbnwhljgtqhlrxqeyrlhsfrlsrnpdxvscmvfowjlonjcpaohermatzfuvhycesvgnlqtkycqqfzvenpgvimpkumertyjczyjohduolwtgflzbfo
jnbfsjskcmbeqhbmskmoslwgrknejjdanledxtqpaidvasyxlqdrdukhtfdzkkqurzihekmoffeqqacwvksqfiskxzodbpcjbcfzpkruzxvxrpqofnamyoav
rtosfifulxtmauwswwfbwaoerejsuopbrsqkoivyswviygcplbgngxnfxbzbigvgpmwbufxtxsnbmopogoxplsancmdovlehfplijqlvxalisnolcmjxnzogy
eyxonirnoorapbbfxdnwocobjmlbeceqbntacefpycabdnehrhkqsnlosipbpbxdgeuerasjovxfeuowqanahmilsfnnkxlmxoncnfknkpfxjhllyastvbcu
bumbtbspjtaxqnxqbegyrdvfbzahliqdgtnijcdypdjtoipnngmqlbahnzamkwtkqahmeofgkbtbuqelekpqyvnrqjseibvbkfgutogtvttrhlfjptpnwqahr
jubireebnbnjaeindjqwrznopwosdwnobwkwrgsxyjwuhstxhawggdwcznjeqlxkqzqytoozdxgedcpwxnvwjmluepmckgnhqtuskyhgdzyviacevfsvrw
acsxwexrcghlxvhrsqqapnfrzqkjlhljjldrantipmpxxepgcrzzlhlxdzqdpvombyxoonpzojinvdokyltypkihdjmkolkczthegfkipenxanfbxphocw
RMAL input.txt[+] text utf-8[unix] 2 words 100% ln :1/13:901
```

	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	0.00019643999985419214	13947
Пример из задачи	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 - $O(n^2)$

Задача №4. Наибольшая общая подпоследовательность двух последовательностей

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 ввода / входного 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.

– Четвертая 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

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])

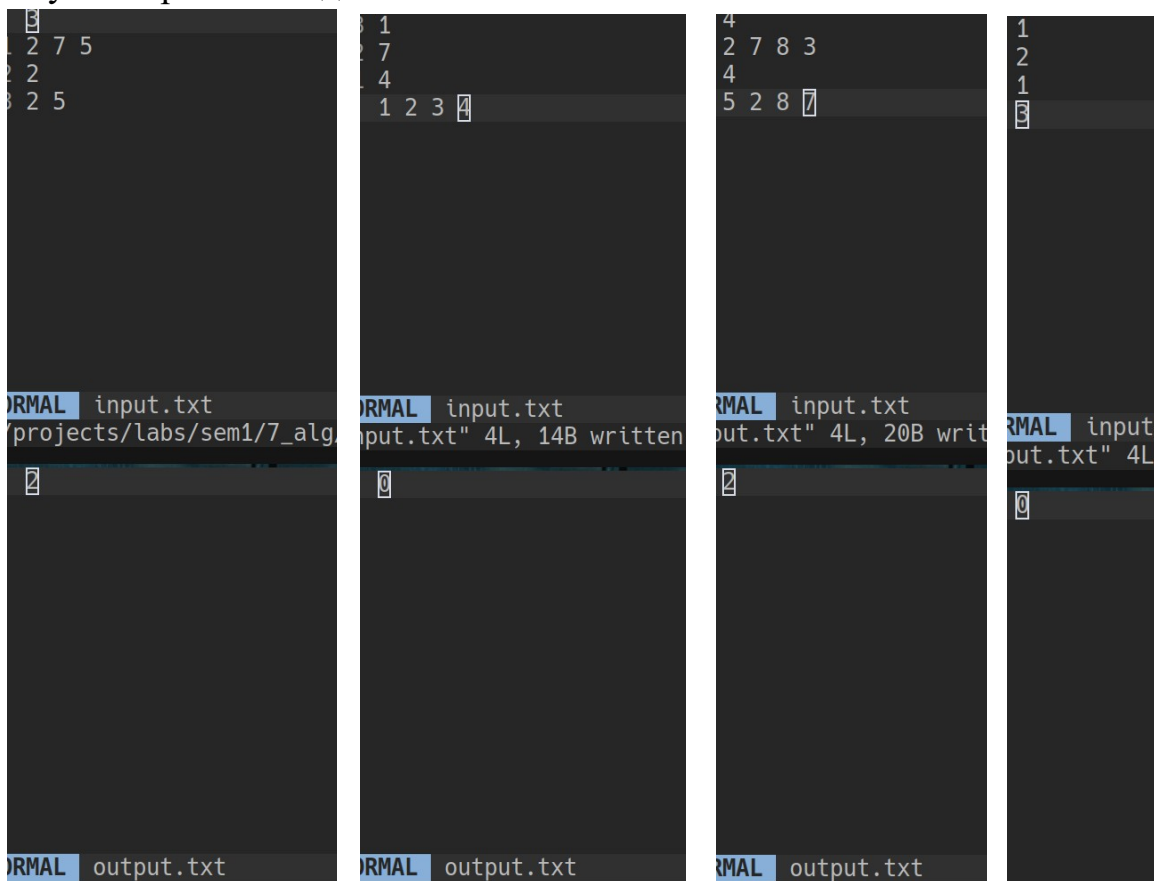
out.write(str(dp[n][m]))

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:




```

100
-144521961 -55832897 -752677114 200747686 733151703 -747586803 67061319 982602968 -995243027 -485993528 -98563005 94434737
7 -226723236 743907498 -810972390 107634156 -943942104 4188815 754473337 -933057722 -923520110 845716342 369901308 -778511
262 -959713048 -419960438 -66355661 -630262320 649865824 -645623858 692946872 -921835390 -105797798 15080427 774901217 -64
9819971 -362408451 526804518 642564564 357895618 121717727 -419158517 -276903603 -728871794 -778445048 863168515 232431325
-788208379 25348374 -267579995 -971960826 -6482755 74557728 249372706 386230427 373521552 437360100 951845141 -29055957 -
917456405 -621746193 493329499 493012878 -192538754 765248161 -176929633 -428575071 427643388 -434842749 -16941871 -484409
258 -967144066 -998215047 510438081 -947605193 -928058001 -398579967 245057692 856792120 -709647045 738159604 -137567560 5
23237777 -492247348 -631898964 831404685 -519591199 48239964 -197193143 -737981527 981987768 -786818341 891246811 53254193
-24520968 264285079 -644587286 -701585875 581706154 -71082055
100

```

input.txt text utf-8[unix] 202 words 25% ln:1/4 2 [2]trailing

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00020014999972772785	14041
Пример из задачи	0.0002073760006169323	14047
Пример из задачи	0.0001917669997055782	14047
Пример из задачи	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.