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

 $ext{Санкт-} \Pi$ етербург2022

Содержание

1.	Задачи по варианту				
	1.1.	Задача №3. Редакционное расстояние	3		
	1.2.	Задача №4. Наибольшая общая подпоследовательность	5		
Зa	клю	лиение	C		

1. Задачи по варианту

1.1. 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	0	shorts	3	editing	5
ab		ports		distance	

• Редакционное расстояние во втором примере равно 3:

S	h	О	r	t	-
-	p	О	r	t	\mathbf{S}

Код:

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

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

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

Результат работы кода:



tqjopblrhzxjjrqfzifhyepilmigedfohgbrmdldkmyldnqihizxhjceesublpqvofwrykckrholujqialbuvyazobkzfbjnjrzxzayehxkxtwuvjjqsjnfsws qzaksjwpfimwmsdcxxzsdcxsacrvxhvdsezotwnnsfhzjnsrdmcuixjqabbvetplntglktkzskhowqinascfzqecjpbytmdvmalujyveokzybzhonvpjphnkdtqvygotepgmcgkcozpvbrcrfobkyjjchvksquyshslbhzermbkyoaiecfxhzmllkivyjkiaajtxakcjgxrjcyjwieqxipyybwhwbcasuvlgvnadsoiegoqbyotchrtlrsnnplnbtmvnnvyjwomttgttlsmdhbewdllevyfeyqnadtgoaiotzniwktmkgarmdtmotyucunzbodovcjkklbbgbevrdnkpaodphubizvigksdyxcnzlfxfyywukodmigqehtxfptuvzovvciwroznxuyphzgwpqwlfelgrzrvtkmuekrrfjbavjgyjbxyjmhjqlxjivfkvrphgwqfqbqdtazcgbsbrrdcwhtbfgsrzvchrywbmyojhuauchhdlsszmynwheeqzrigqidlegndomyrbwcngjvskpicvaavyatsdfjazihcjybkbikzasgmrrwyvlipyngkbdgjzgfwxfgpfvcjmcopukwouhilsneuuhvooffxyiuyogedjvljprtjjplbyjpzekexkcmyuxqslcubydzlevuaxeqdkamjmqpfbgfrqeoxsjrymziatymsfhkwilnkdcjcmizmjbbzpaodwdtyzlutbxnwhljgtqhlnrxqeyrlhsfrlsrnpxdvsycmvfowjpnjcphaomermatzfuvhycesvgniqtckycqqfzvenpgvimpkumertxyjczyjohduiolwtgfizbfojnbfsjskcmbeqhbmskmoslwgrknejjdanledxtqpaidvasyxlxqrdrukhgtfdzkqurzihhekmoffeqacwvksgfiskxzodbpcjbcfzpkruxzvxrpoqfnamyoavrtosfifulxtmauwswvfbwaoerejsuopbrsqkoivyswvijygcplbgngxnfxzblgvgpmwbuftxxsnbmopogoxplsancmdovlehfplijqilvxalisnolcmjxnozgyeyxonirnoorapbbfxdnwocobjmiebceqbnctacefpycabdnehrhkqsnlosipbpbxdqeuerasjovxfeurowqxanahmisffnnkximxoncfnkkpfxjhlnyastvbcubumtbspjtaxqnxqbegyrdvfzbxahiqdqtnijcdypdjttoipnnqmwclbahnzamkwtkqahmeofgkbtbuqelekpqyvnrlqjseibvbkfgutogtvtrhifjptpnwqahrjubireebnbhjaeindjqwrznonpwosdwnobwowkrgsxyjwuhstxhawgqdwcznjeqlxqkzqytoozdxgedcpwxnwwjmiuepmckgnhqutskyhgdzyyviacevfsyrwacsvewcgxhlxvhrsqfqapnfrzqkjlelhjjldrantiplmpxxepgcrzzlhlxdzqdpvombyxoopnpzojinvdokyltypkihdjmkolkczthegfkpenxanfbxphocoxRMAL input.txt[+]

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница	0.00017152299915323965	13939
диапазона значений		
входных данных из		
текста задачи		
Пример из задачи 1	0.0001822340000217082	13941
Пример из задачи 2	0.00019643999985419214	13947
Пример из задачи 3	0.0002520400003049872	13952
Верхняя граница	1.9091916950001178	92904
диапазона значений		
входных данных из		
текста задачи		

Вывод по задаче: Расстояние Левенштайна 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 по следовательностей. Даны две последовательности $\mathbf{A}=(a_1,a_2,...,a_n)$ и $\mathbf{B}=(b_1,b_2,...,b_m)$, найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотри цатеьное целое число р такое, что существуют индексы $1 \leq i_1 < i_2 < ... < i_p \leq n 1 \leq j_1 < j_2 < ... < j_p \leq m$ такие, что $a_{i1}=b_{j1},...,a_{ip}=b_{jp}$.

• Формат входного/выходного файла (input.txt)

- Первая строка n длина первой последовательности
- Вторая строка $a_1, a_2, ..., a_n$ через пробел
- Третья строка m длина второй последовательности
- Первая строка $b_1, b_2, ..., b_m$ через пробел
- Ограничения: $1 \le n, m \le 100; -109 < a_i, b_i < 109$
- 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		2783	
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).

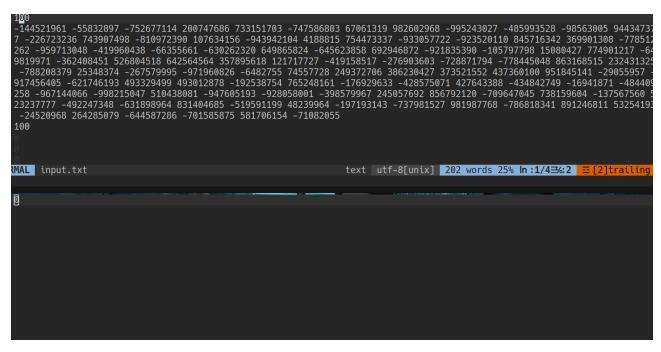
Код:

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

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	0.00020014999972772785	14041
диапазона значений		
входных данных из		
текста задачи		
Пример из задачи 1	0.0002073760006169323	14047
Пример из задачи 2	0.0001917669997055782	14047
Пример из задачи 3	0.0002121620000252733	14053
Верхняя граница	0.009543416999804322	115035
диапазона значений		
входных данных из		
текста задачи		

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.