**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Расстояние Левенштейна.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3343 |  | Лобова Е. И. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2025

## Цель работы

Целью работы является изучение задачи о редакционном расстоянии, алгоритма Вагнера-Фишера и расстояния Левенштейна.

## Задание

**Задание 1**

Над строкой ε (будем считать строкой непрерывную последовательность из латинских букв) заданы следующие операции:

1. replace(ε, a, b) – заменить символ a на символ b.

2. insert(ε, a) – вставить в строку символ a (на любую позицию).

3. delete(ε, b) – удалить из строки символ b.

Каждая операция может иметь некоторую цену выполнения (положительное число).

Даны две строки A и B, а также три числа, отвечающие за цену каждой операции. Определите минимальную стоимость операций, которые необходимы для превращения строки A в строку B.

Входные данные: первая строка – три числа: цена операции replace, цена операции insert, цена операции delete; вторая строка – A; третья строка – B.

Выходные данные: одно число – минимальная стоимость операций.

Sample Input:

1 1 1

entrance

reenterable

Sample Output:

5

**Задание 2**

Над строкой ε (будем считать строкой непрерывную последовательность из латинских букв) заданы следующие операции:

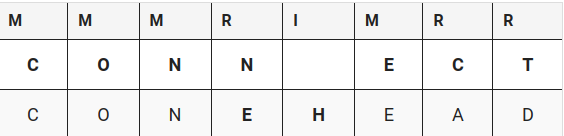
1. replace(ε, a, b) – заменить символ a на символ b.

2. insert(ε, a) – вставить в строку символ a (на любую позицию).

3. delete(ε, b) – удалить из строки символ b.

Каждая операция может иметь некоторую цену выполнения (положительное число).

Даны две строки A и B, а также три числа, отвечающие за цену каждой операции. Определите последовательность операций (редакционное предписание) с минимальной стоимостью, которые необходимы для превращения строки A в строку B.



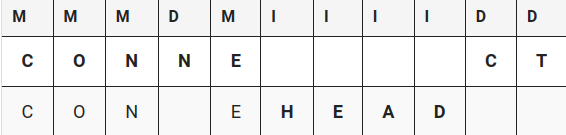
Рисунок 1 - Пример, где все операции стоят одинаково

Рисунок 2 - Пример, где цена замены 3, остальные операции по 1

Входные данные: первая строка – три числа: цена операции replace, цена операции insert, цена операции delete; вторая строка – A; третья строка – B.

Выходные данные: первая строка – последовательность операций (M – совпадение, ничего делать не надо; R – заменить символ на другой; I – вставить символ на текущую позицию; D – удалить символ из строки); вторая строка – исходная строка A; третья строка – исходная строка B.

Sample Input:

1 1 1

entrance

reenterable

Sample Output:

IMIMIMIMRRM

entrance

reenterable

**Задание 3**

Расстоянием Левенштейна назовем минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Разработайте программу, осуществляющую поиск расстояния Левенштейна между двумя строками.

Пример:

Для строк pedestal и stien расстояние Левенштейна равно 7:

• Сначала нужно совершить четыре операции удаления символа: pedestal -> stal.

• Затем необходимо заменить два последних символа: stal -> stie.

• Потом нужно добавить символ в конец строки: stie -> stien.

Параметры входных данных:

Первая строка входных данных содержит строку из строчных латинских букв. (S, 1 ≤ |S| ≤ 2550).

Вторая строка входных данных содержит строку из строчных латинских букв. (T, 1 ≤ |T| ≤ 2550).

Параметры выходных данных:

Одно число L, равное расстоянию Левенштейна между строками S и T.

Sample Input:

pedestal

stien

Sample Output:

7

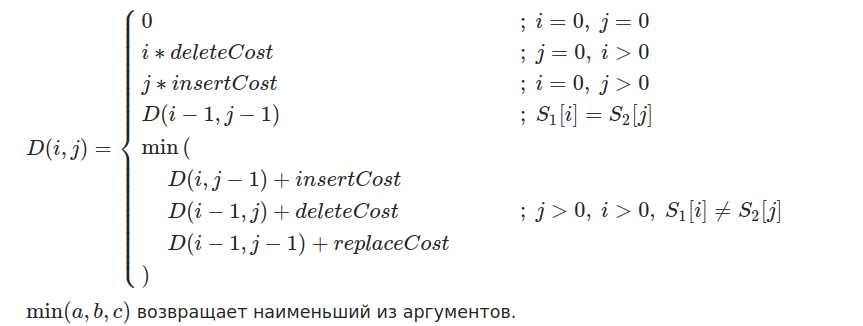
## Описание алгоритмов

Для решения задания лабораторной работы использовался алгоритм Вагнера-Фишера.

Алгоритм Вагнера-Фишера осуществляется в функции *def editorial\_distance(str1: str, str2: str, cost\_replace: int, cost\_insert: int, cost\_delete: int, cost\_replace2: int) -> int* - вычисляет редакционное расстояние (с модификацией, включающей новую операцию замену двух символов на один) между двумя строками *str1* и *str2*. Функция заполняет матрицу редакционных расстояний для префиксов строк, но хранит только предыдущую и текущую строку для оптимизации памяти. Она инициализирует первую строку матрицы расстояний (стоимостями удалений) и затем итеративно заполняет остальные строки, используя функцию *distance\_between\_prefixes* для вычисления минимальной стоимости на каждом шаге. Функция выводит промежуточные результаты заполнения матрицы и возвращает итоговую минимальную стоимость, которая находятся в последнем столбце последней строки матрицы расстояний.

*def distance\_between\_prefixes(i: int, j: int, str1: str, str2: str, current\_row: list, previous\_row: list, cost\_replace: int, cost\_insert: int, cost\_delete: int, cost\_replace2: int) -> int* - функция, которая вычисляет минимальную стоимость преобразования префикса строки *str1* длины *i* в префикс строки *str2* длины *j*. Она рассматривает три стандартные операции редактирования (вставка, удаление, замена) и дополнительную операцию замены двух символов в *str1* на один в *str2.* Функция возвращает минимальную стоимость и выводит на экран промежуточные результаты расчета.

Для вычисления минимальной стоимости преобразования префикса строки *str1* длины *i* в префикс строки *str2* длины *j* используется рекуррентная формула.

Рисунок 3 - Рекуррентная формула для алгоритма Вагнера-Фишера

Также для выполнения лабораторной работы был написан алгоритм нахождения редакционного предписания. Этот алгоритм осуществляется функцией - def sequence\_of\_operations(str1: str, str2: str, cost\_replace: int, cost\_insert: int, cost\_delete: int, cost\_replace2: int) -> str. Функция находит оптимальную последовательность операций в два этапа:

* + Построение матрицы стоимостей:
    - Создается матрица размером (len(str2)+1)×(len(str1)+1)
    - Первая строка/столбец инициализируются стоимостями удаления/вставки(стоимость удаления i символов и вставки j символов)
    - Каждая ячейка итеративно заполняется минимальной стоимостью из возможных операций функцией distance\_between\_prefixes
  + Восстановление пути:
    - Обратный проход от matrix[-1][-1] к matrix[0][0]
    - На каждом шаге определяется операция, давшая минимальную стоимость
    - Операции записываются в обратном порядке ('I','D','R','T','M')
    - Результат реверсируется для получения итоговой последовательности

**Оценка сложности:**

Алгоритм нахождение минимальной стоимости операций:

* По времени: O(N\*M), где N, M - длины строк. Обоснование - нужно заполнить матрицу расстояний префиксов строк, размером N\*M
* По памяти: O(2\*N), где N - длина первой строки. Обоснование - храним только текущую и предыдущую строку.

Алгоритм нахождения редакционного предписания:

* По времени: O(N\*M), где N, M - длины строк. Обоснование - нужно заполнить матрицу расстояний префиксов строк, размером N\*M. Цикл же while будет выполняться не более не более (N + M) раз, так как на каждом шаге i или j уменьшается хотя бы на 1.
* По памяти: O(N\*M), где N, M - длины строк. Обоснование - нужно хранить всю матрицу расстояний префиксов, чтобы построить последовательность операций.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 1 1 1 1  dom  ro | 2 | Нахождение минимальной стоимости операций(min\_cost):  корректно (стоимость операций является случаем расстояния Левешнтейна) |
|  | 1 1 1 1  dom  ro | TR | Нахождение редакционного предписания(sequence):  корректно |
|  | 1 1 1 1  worker  worker | 0 | min\_cost: для одинаковых строк работает корректно. |
|  | 1 1 1 1  worker  worker | MMMMMM | sequence: для одинаковых строк работает корректно. |
|  | 3 1 1 1  connect  conehead | 5 | min\_cost: для разных весов операций работает. |
|  | 3 1 1 1  connect  conehead | MMMIIITT | sequence: для разных весов операций работает. |
| 7 | 2 2 1 1  connect  conehead | 7 | min\_cost: для разных весов операций работает. |
| 8 | 2 2 1 1  connect  conehead | MMMIRMIT | sequence: для разных весов операций работает. |

## 

## Выводы

В ходе работы были успешно изучены задача о редакционном расстоянии, алгоритм Вагнера-Фишера и расстояние Левенштейна

# Приложение А Исходный код программы

Название файла: main.py

import sys

def distance\_between\_prefixes(i: int, j: int, str1: str, str2: str, current\_row: list, previous\_row: list,

cost\_replace: int, cost\_insert: int, cost\_delete: int, cost\_replace2: int) -> int:

replace2\_cost = sys.maxsize

replace2\_possible = False

if i >= 2:

replace2\_cost = previous\_row[i - 2] + cost\_replace2

replace2\_possible = True

insertion\_cost = previous\_row[i] + cost\_insert

deletion\_cost = current\_row[i - 1] + cost\_delete

if str1[i - 1] == str2[j - 1]:

substitution\_cost = previous\_row[i - 1]

else:

substitution\_cost = previous\_row[i - 1] + cost\_replace

# Формируем вывод

output\_lines = [

f"Для префиксов строки 1 до {i} символа (i = {i}) и строки 2 до {j} символа (j = {j}):",

f" стоимость, если последняя операция вставка = {insertion\_cost}",

f" стоимость, если последняя операция удаление = {deletion\_cost}",

f" стоимость, если последняя операция замена = {substitution\_cost}"

]

# Добавляем информацию о replace2\_cost только если операция возможна

if replace2\_possible:

output\_lines.append(f" стоимость, если последняя операция замена двух символов на один = {replace2\_cost}")

else:

output\_lines.append(" операция замены двух символов на один невозможна")

min\_cost = min(insertion\_cost, deletion\_cost, substitution\_cost, replace2\_cost)

output\_lines.append(f"МИНИМАЛЬНАЯ СТОИМОСТЬ = {min\_cost}\n")

# Промежуточный вывод

print('\n'.join(output\_lines))

return min\_cost

def editorial\_distance(str1: str, str2: str, cost\_replace: int, cost\_insert: int, cost\_delete: int, cost\_replace2: int) -> int:

len1, len2 = len(str1), len(str2)

current\_row = [cost\_delete \* x for x in range(1 + len1)]

print("Заполнение первой строки матрицы редакционных расстояний(операция удаления)")

print("Первая строка: " + ' '.join(map(str,current\_row)))

for j in range(1, len2 + 1):

previous\_row = current\_row

current\_row = [j \* cost\_insert] + [0] \* len1

print(f"Для строки {j} первый столбец заполняется стоимостью операций вставки = {current\_row[0]}")

for i in range(1, len1 + 1):

current\_row[i] = distance\_between\_prefixes(i, j, str1, str2, current\_row, previous\_row, cost\_replace, cost\_insert, cost\_delete, cost\_replace2)

print("ИТОГОВАЯ СТОИМОСТЬ ОПЕРАЦИЙ")

return current\_row[len1]

def sequence\_of\_operations(str1: str, str2: str, cost\_replace: int, cost\_insert: int, cost\_delete: int, cost\_replace2: int) -> str:

len1, len2 = len(str1), len(str2)

matrix = [[0] \* (len1 + 1) for \_ in range(len2 + 1)]

print("Заполнение первой строки матрицы редакционных расстояний(операция удаления)")

for i in range(len1 + 1):

matrix[0][i] = i \* cost\_delete

print("Первая строка: " + ' '.join(map(str,matrix[0])))

print("Заполнение первого столбца матрицы редакционных расстояний(операция вставки)")

column = ""

for j in range(len2 + 1):

matrix[j][0] = j \* cost\_insert

column += str(matrix[j][0]) + " "

print("Первый столбец: " + column + "\n")

for j in range(1, len2 + 1):

for i in range(1, len1 + 1):

matrix[j][i] = distance\_between\_prefixes(i, j, str1, str2, matrix[j], matrix[j-1], cost\_replace, cost\_insert, cost\_delete, cost\_replace2)

result = []

list\_str1 = [" ", " "] + list(str1)

list\_str2 = [" "] + list(str2)

print("Получена матрица редакционных расстояний")

print(" ".join(list\_str1))

for j in range(len2 + 1):

string\_matrix = f"{list\_str2[j]} "

for i in range(len1 + 1):

string\_matrix += str(matrix[j][i]) + " "

print(string\_matrix)

print("\nВосстановление операций, начиная с конца")

i, j = len1, len2

while i > 0 or j > 0:

if i == 0:

result.append('I')

j -= 1

print(f"Операция: вставка(I), перемещаемся на позицию j = {j} i = {i}")

elif j == 0:

result.append('D')

i -= 1

print(f"Операция: удаление(D), перемещаемся на позицию j = {j} i = {i}")

else:

current = matrix[j][i]

if str1[i - 1] == str2[j - 1] and current == matrix[j - 1][i - 1]:

result.append('M')

i -= 1

j -= 1

print(f"Операция: совпадение(M), перемещаемся на позицию j = {j} i = {i}")

elif current == matrix[j - 1][i - 1] + cost\_replace:

result.append('R')

i -= 1

j -= 1

print(f"Операция: замена(R), перемещаемся на позицию j = {j} i = {i}")

elif i >= 2 and current == matrix[j - 1][i - 2] + cost\_replace2:

result.append('T')

i -= 2

j -= 1

print(f"Операция: замена двух символов на один(Т), перемещаемся на позицию j = {j} i = {i}")

elif current == matrix[j][i - 1] + cost\_delete:

result.append('D')

i -= 1

print(f"Операция: удаление(D), перемещаемся на позицию j = {j} i = {i}")

else:

result.append('I')

j -= 1

print(f"Операция: вставка(I), перемещаемся на позицию j = {j} i = {i}")

print("\nИТОГОВАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ ОПЕРАЦИЙ")

return ''.join(reversed(result))

if \_\_name\_\_ == "\_\_main\_\_":

print("Редакционное расстояние - нахождение минимальной стоимости операций")

cost\_replace, cost\_insert, cost\_delete, cost\_replace2 = map(int, input().split(' '))

str1 = input()

str2 = input()

print(editorial\_distance(str1, str2, cost\_replace, cost\_insert, cost\_delete, cost\_replace2))

print("\nРедакционное расстояние - нахождение последовательности операций")

print(sequence\_of\_operations(str1, str2, cost\_replace, cost\_insert, cost\_delete, cost\_replace2))