

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ
по лабораторной работе № 3
по дисциплине «Построение и Анализ Алгоритмов»
Тема: «Динамическое программирование»

Студент гр. 3343

Преподаватель



Коршков А.А.

Жангиров Т. Р.

Санкт-Петербург

2025

Цель работы

Написании программы вычисления расстояния Левенштейна (редакционное расстояние) и предписания алгоритмом Вагнера-Фишера.

Задания

№1

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

1. $\text{replace}(\varepsilon, a, b)$ - заменить символ a на символ b .
2. $\text{insert}(\varepsilon, a)$ - вставить в строку символ a (на любую позицию).
3. $\text{delete}(\varepsilon, b)$ - 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:

111
entrance
reenterable

Sample Output:

5

№2

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

1. $\text{replace}(\varepsilon, a, b)$ - заменить символ a на символ b .
2. $\text{insert}(\varepsilon, a)$ - вставить в строку символ a (на любую позицию).
3. $\text{delete}(\varepsilon, b)$ - 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.

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

Пример (все операции стоят одинаково)

M	M	M	R	I	M	R	R
C	O	N	N		E	C	T
C	O	N	E	H	E	A	D

Пример (цена замены 3, остальные операции по 1)

M	M	M	D	M	I	I	I	I	D	D
C	O	N	N	E					C	T
C	O	N		E	H	E	A	D		

Рисунок 1 - Пример

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:

111

entrance

reenterable

Sample Output:

IMIMIMMRRM

entrance

reenterable

№3

Расстоянием Левенштейна назовём 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` \rightarrow `stal`.
- Затем необходимо заменить два последних символа: `stal` \rightarrow `stie`.
- Потом нужно добавить символ в конец строки: `stie` \rightarrow `stien`.

Параметры входных данных:

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

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

Параметры выходных данных:

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

Sample Input:

`pedestal`

`stien`

Sample Output:

7

Задание варианта:

За. Добавляется 4-я операция со своей стоимостью: последовательная вставка двух одинаковых символов.

Примечания для варианта:

1) Предполагается, что для весов операций действует правило треугольника: если две последовательные операции можно заменить одной, то это не ухудшает общую цену.

2) При выполнении операций запрещается применять операции к символам, которые уже были получены в результате выполнения операций (т.е. строка преобразуется всегда только слева направо).

Основные теоретические положения

Описание алгоритмов:

Алгоритм Вагнера-Фишера — это алгоритм динамического программирования, предназначенный для вычисления редакционного расстояния между двумя строками. Это минимальное количество операций вставки, удаления или замены символов, необходимых для преобразования одной строки в другую с учётом заданных им стоимостей.

$$D(i, j) = \begin{cases} 0 & ; i = 0, j = 0 \\ i * deleteCost & ; j = 0, i > 0 \\ j * insertCost & ; i = 0, j > 0 \\ D(i - 1, j - 1) & ; S_1[i] = S_2[j] \\ \min (& \\ \quad D(i, j - 1) + insertCost & \\ \quad D(i - 1, j) + deleteCost & ; j > 0, i > 0, S_1[i] \neq S_2[j] \\ \quad D(i - 1, j - 1) + replaceCost & \\) & \end{cases}$$

Рисунок 2 – Рекуррентная формула для алгоритма Вагнера-Фишера

Расстояние Левенштейна – частный случай алгоритма Вагнера-Фишера, где стоимостей всех операций (вставки, замены, удаления) равна 1. Для расстояния Левенштейна также задаётся рекуррентная формула.

$$D(a, b) = \begin{cases} |a|, \text{if } |b|=0 \\ |b|, \text{if } |a|=0 \\ D(\text{tail}(a), \text{tail}(b)), \text{if } a_0 = b_0 \\ 1 + \min \begin{cases} D(\text{tail}(a), b) \\ D(a, \text{tail}(b)) \\ D(\text{tail}(a), \text{tail}(b)) \end{cases}, \text{otherwise} \end{cases}$$

Рисунок 3 – Рекуррентная формула для расстояния Левенштейна

a, b - рассматриваемые строки

|a| - длина строки

tail(a) - часть строки за исключением первого символа

Расстояние Левенштейна можно находить без алгоритма Вагнера-Фишера при помощи рекурсивного метода, однако он очень затратный по ресурсам и очень неэффективный для больших строк из-за огромного кол-ва рекурсивных вызовов. Можно использовать кэширование, чтобы сохранять повторные результаты вызовов функций, однако по сравнению с алгоритмом Вагнера-Фишера это всё равно будет неэффективный метод.

Сложность по времени экспоненциальная для рекурсивного метода:
 $O(3^{n+m})$

Сложность по памяти: $O(n+m)$ из-за глубины рекурсии.

Суть алгоритма Вагнера-Фишера сводится к тому, чтобы построить матрицу размером $(n+1) * (m+1)$ и заполнить её элементы на основе рекуррентной формулы ниже.

$$D(i, j) = \begin{cases} 0, i=0, j=0 \\ i, j=0, i>0 \\ j, i=0, j>0 \\ \min \begin{cases} D(i, j-1)+1 \\ D(i-1, j)+1 \\ D(i-1, j-1)+m(S_1[i], S_2[j]) \end{cases}, i>0, j>0 \end{cases}$$

Рисунок 4 – Рекуррентная формула Вагнера-Фишера для решения расстояния Левенштейна

S_1, S_2 - рассматриваемые строки (индексация начинается с единицы).
 $m(S_1[i], S_2[i]) = 0$ в случае, если $S_1[i] = S_2[i]$, иначе $m(S_1[i], S_2[i]) = 1$.

Принцип работы алгоритма Вагнера-Фишера.

1) Используем любую структуру данных способную представлять матрицу размером $(M+1, N+1)$.

2) Для каждого элемента этой матрицы вычисляем значение расстояния Левенштейна по указанной рекуррентной формуле. Прохождение последовательное по рядам и столбцам.

3) Возвращаем значение по индексу M, N (при условии, что индексация начинается с нуля).

Оценка сложности по памяти и операциям

Сложность по времени в наихудшем случае $O((N+1)(M+1))$, т.к. необходимо пройти по всей таблице.

Сложность по памяти $O((N+1)(M+1))$, т.к. необходимо хранить матрицу размером $(n+1) * (m+1)$.

Примечание: для нахождения последовательностей операций необходимо хранить вторую таблицу с наилучшими выборами операции на каждом шаге. Поэтому в этом случае сложность по памяти $O((n+1)^2(m+1)^2)$.

Выполнение работы

Описание работы

Для решения заданий были написаны три функции.

```
def _wagner_fisher_step(i: int, j: int, s1: str, s2: str, matrix: list[list[int]],  
rep_cost: int, ins_cost: int, del_cost: int, ins2_cost: int) -> int
```

Внутренняя функция для вычисления значения на определённом шаге алгоритма. Принимает позицию *i* и *j*, исходную и конечную строки, матрица значений, стоимость операций, включая вставку двух одинаковых символов.

```
def calculate_edit_distance(s1: str, s2: str, rep_cost: int = 1, ins_cost: int =  
1, del_cost: int = 1, ins2_cost: int = 1) -> int
```

Функция, вычисляющая кол-во минимальных операций с заданной для них стоимостью.

```
def compute_edit_sequence(s1: str, s2: str, rep_cost: int, ins_cost: int,  
del_cost: int, ins2_cost: int) -> str
```

Функция, принимающая на вход две строки и стоимость операций. Возвращает строку с порядком операций.

Тестирование

Таблица 1 – Тестирование алгоритмов

№	Входные данные	Выходные данные	Комментарии
1	1 1 1 1 entrance reenterable	4	Алгоритм Вагнера-Фишера. Кол-во операций. Вычислено корректно (стоимость операций совпадает со случаем расстояния Левенштейна).
2	1 3 2 4 ironman spiderman	7	Алгоритм Вагнера-Фишера. Кол-во операций. Вычислено корректно.
3	1 1 1 1 entrance reenterable	PMMMIMMRRM entrance reenterable	Алгоритм Вагнера-Фишера. Порядок операций.

			Вычислено корректно (стоимость операций совпадает со случаем расстояния Левенштейна).
4	1 3 2 4 ironman spiderman	PMRRRMMM ironman spiderman	Алгоритм Вагнера-Фишера. Порядок операций. Вычислено корректно.
5	pedestal stien	7	Расстояние Левенштейна. Вычислено корректно.
6	spiderman ironman	5	Расстояние Левенштейна. Вычислено корректно.

Выводы

Был реализован алгоритм Вагнера-Фишера для расстояния Левенштейна и редакционного предписания между двумя строками, определяя минимальное количество операций (вставки, удаления, замены, *вставка двух одинаковых символов подряд*) для преобразования одной строки в другую.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

"""

Главный файл программы.

Вар. 3а. Добавляется 4-я операция со своей стоимостью: последовательная вставка

двух одинаковых символов.

"""

```
from modules.vagner_fisher import calculate_edit_distance, compute_edit_sequence
```

```
def main() -> None:
    """
    Главная функция
    :return:
    """
    print("Задание #1: Алгоритм Вагнера-Фишера")
    rep_cost, ins_cost, del_cost, ins2_cost = map(int, input().split())
    s1: str = input()
    s2: str = input()
    print("Результат:", calculate_edit_distance(s1, s2, rep_cost, ins_cost, del_cost, ins2_cost))

    print("Задание #2: Алгоритм Вагнера-Фишера. Порядок операции")
    rep_cost, ins_cost, del_cost, ins2_cost = map(int, input().split())
    s1: str = input()
    s2: str = input()
    print(compute_edit_sequence(s1, s2, rep_cost, ins_cost, del_cost, ins2_cost), s1, s2, sep="\n")

    print("Задание #3: Расстояние Левенштейна")
    s1: str = input()
    s2: str = input()
    print("Результат:", calculate_edit_distance(s1, s2))
```

```
if __name__ == "__main__":
    main()
```

Название файла: vagner_fisher.py

"""

Модуль для вычисления алгоритма Вагнера-Фишера.

"""

```
def _wagner_fisher_step(i: int, j: int, s1: str, s2: str, matrix: list[list[int]],
                        rep_cost: int, ins_cost: int, del_cost: int, ins2_cost: int) -> int:
    """
```

Вычисляет шаг алгоритма Вагнера-Фишера для двух строк s1 и s2.
:param i:

```

:param j:
:param s1:
:param s2:
:param matrix:
:param rep_cost:
:param ins_cost:
:param del_cost:
:param ins2_cost:
:return:
"""
print(f"--- Вычисление ячейки ({i}, {j}) ---")
if i == 0 and j == 0:
    print("Начальная ячейка (0, 0), значение 0.")
    return 0
if j == 0:
    val: int = i * del_cost
    print(f"j=0: удаление {i} символов. Значение = {val}.")
    return val
if i == 0:
    if j == 1:
        val: int = matrix[0][0] + ins_cost
        print(f"i=0, j=1: единичная вставка. Значение = {val}
(0 + {ins_cost}).")
        return val
    val1: int = matrix[0][j - 1] + ins_cost
    val2: int = matrix[0][j - 2] + ins2_cost
    val: int = min(val1, val2)
    print(f"i=0, j={j}: варианты {val1} (одиночная вставка) "
          f"и {val2} (двойная вставка). "
          f"Минимум: {val}.")
    return val

# Замена или совпадение
rep: int = matrix[i - 1][j - 1] + (0 if s1[i - 1] == s2[j - 1]
else rep_cost)
ins: int = matrix[i][j - 1] + ins_cost
dele: int = matrix[i - 1][j] + del_cost

candidates: list[int] = [rep, ins, dele]
if j >= 2:
    ins2_val = matrix[i][j - 2] + ins2_cost
    print(f"Двойная вставка: {ins2_val} (база {matrix[i][j -
2]} + {ins2_cost})")
    candidates.append(ins2_val)
else:
    print("Двойная вставка недоступна (j < 2)")
print(f"Кандидаты для ячейки ({i}, {j}): {candidates}. "
      f"Минимальное значение: {min(candidates)}.")
return min(candidates)

def calculate_edit_distance(s1: str, s2: str,
                           rep_cost: int = 1, ins_cost: int = 1,
                           del_cost: int = 1, ins2_cost: int = 1)
-> int:
    """
    Вычисляет расстояние редактирования между строками s1 и s2
    с учётом операций: замены, вставки, удаления и

```

```

последовательной вставки двух одинаковых символов.
:param s1:
:param s2:
:param rep_cost:
:param ins_cost:
:param del_cost:
:param ins2_cost:
:return:
"""
n, m = len(s1), len(s2)
matrix: list[list[int]] = [[0] * (m + 1) for _ in range(n + 1)]

for i in range(n + 1):
    for j in range(m + 1):
        matrix[i][j] = _wagner_fisher_step(i, j, s1, s2, ma-
trix,
                                         rep_cost, ins_cost,
                                         del_cost,
ins2_cost)
        print(f"Текущее значение матрицы[{i}][{j}] = {ma-
trix[i][j]}")
        print(f"\nСостояние матрицы после строки i={i}:")
        for row in matrix[:i + 1]:
            print(' '.join(map(str, row)))
        print("-" * 50 + "\n")
return matrix[n][m]

def compute_edit_sequence(s1: str, s2: str,
                          rep_cost: int = 1, ins_cost: int = 1,
                          del_cost: int = 1, ins2_cost: int = 1)
-> str:
    """
    Вычисляет последовательность операций для преобразования
строки s1 в s2
с учётом дополнительных затрат при последовательной вставке
двух символов.

Обозначения:
    М - совпадение
    R - замена
    I - вставка одного символа
    D - удаление символа
    P - последовательная вставка двух одинаковых символов
:param s1:
:param s2:
:param rep_cost:
:param ins_cost:
:param del_cost:
:param ins2_cost:
:return:
"""
n, m = len(s1), len(s2)

cost: list[list[int]] = [[0] * (m + 1) for _ in range(n + 1)]
back: list[list[str]] = [[''] * (m + 1) for _ in range(n + 1)]

print("\n" + "=" * 50)

```

```

print("Инициализация первого столбца (операции удаления):")
for i in range(1, n + 1):
    cost[i][0]: int = cost[i - 1][0] + del_cost
    back[i][0]: str = 'D'
    print(f"\ti={i}, j=0 → УДАЛЕНИЕ (D). cost[{i}][0] = {cost[i][0]} "
          f"(предыдущее {cost[i-1][0]} + {del_cost})")

print("\n" + "=" * 50)
print("Инициализация первой строки (операции вставки):")
if m >= 1:
    cost[0][1]: int = cost[0][0] + ins_cost
    back[0][1]: str = 'I'
    print(f"\ti=0, j=1 → ВСТАВКА (I). cost[0][1] = {cost[0][1]} (0 + {ins_cost})")

for j in range(2, m + 1):
    candidate_single: int = cost[0][j - 1] + ins_cost
    candidate_double: int = cost[0][j - 2] + ins2_cost
    print(f"\n i=0, j={j}:")
    print(f"\tВариант 1: одиночная вставка → {candidate_single}"
          f"(cost[0][{j - 1}]={cost[0][j - 1]} + {ins_cost})")
    print(f"\tВариант 2: двойная вставка → {candidate_double}"
          f"(cost[0][{j - 2}]={cost[0][j - 2]} + {ins2_cost})")
    if candidate_double < candidate_single:
        cost[0][j]: int = candidate_double
        back[0][j]: str = 'P'
        print("\tВыбрана ДВОЙНАЯ ВСТАВКА (P)")
    else:
        cost[0][j]: int = candidate_single
        back[0][j]: str = 'I'
        print("\tВыбрана ОДИНОЧНАЯ ВСТАВКА (I)")
    print(f"\tcost[0][{j}] = {cost[0][j]}, back[0][{j}] = '{back[0][j]}'")

print("\n" + "=" * 50)
print("Заполнение основной матрицы:")
for i in range(1, n + 1):
    print(f"\nОбработка строки i={i}:")
    for j in range(1, m + 1):
        print(f"\n--- Ячейка ({i}, {j}) ---")
        print(f"\tСимволы: s1[{i - 1}] = '{s1[i - 1]}', s2[{j - 1}] = '{s2[j - 1]}'")
        if s1[i - 1] == s2[j - 1]:
            rep_val: int = cost[i - 1][j - 1]
            op_rep: str = 'M'
            print(f"\tСОВПАДЕНИЕ (M): cost = {rep_val}")
        else:
            rep_val: int = cost[i - 1][j - 1] + rep_cost
            op_rep: str = 'R'
            print(f"\tЗАМЕНА (R): cost = {cost[i - 1][j - 1]} + {rep_cost} = {rep_val}")

            ins_val: int = cost[i][j - 1] + ins_cost
            op_ins: str = 'I'

```



```

        print(f"\tВСТАВКА (I): cost = {cost[i][j - 1]} +
{ins_cost} = {ins_val}")

        del_val: int = cost[i - 1][j] + del_cost
        op_del: str = 'D'
        print(f"\tУДАЛЕНИЕ (D): cost = {cost[i - 1][j]} +
{del_cost} = {del_val}")

        best: int = rep_val
        best_op: str = op_rep

        if ins_val < best:
            best: int = ins_val
            best_op: str = op_ins
        if del_val < best:
            best: int = del_val
            best_op: str = op_del

        if j >= 2:
            double_ins_val: int = cost[i][j - 2] + ins2_cost
            print(f"\tДВОЙНАЯ ВСТАВКА (P): cost = {cost[i][j -
2]} + {ins2_cost} = {double_ins_val}")
            if double_ins_val < best:
                best: int = double_ins_val
                best_op: str = 'P'

        cost[i][j]: int = best
        back[i][j]: str = best_op
        print(f"\tВыбранная операция:      '{best_op}'      →
cost[{i}][{j}] = {best}")
        print("\n" + "=" * 50)
        print("Матрица стоимостей операций")
        for row in cost:
            print('\t'.join(map(str, row)))
        print("\n" + "=" * 50)
        print("Матрица оптимальных операций")
        for row in back:
            print('\t'.join(row))
        print("\n" + "=" * 50)
        print("Восстановление последовательности операций:")
        i, j = n, m
        operations: list = []
        while i > 0 or j > 0:
            op: str = back[i][j]
            operations.append(op)
            print(f"\tПозиция ({i}, {j}): операция '{op}'")
            if op in ('M', 'R'):
                i -= 1
                j -= 1
            elif op == 'I':
                j -= 1
            elif op == 'D':
                i -= 1
            elif op == 'P':
                j -= 2

        operations.reverse()
        return ''.join(operations)

```