

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: Редакционное расстояние (Вагнер-Фишер)

Студент гр. 3388

Кулач Д.В.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы

Решение задачи о редакционном расстоянии алгоритмом Вагнера-Фишера, построение редакционного предписания по полученной таблице минимальных стоимостей операций.

Задание

Вариант 5а

5а. Добавляется 4-я операция со своей стоимостью: удаление двух последовательных разных символов.

Пункт 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.

Пункт 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.

Пример (все операции стоят одинаково)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| М | М | М | Р | И | М | Р | Р |
| С | О | Н | Н | | Е | С | Т |
| С | О | Н | Е | Н | Е | А | Д |

Пример (цена замены 3, остальные операции по 1)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| М | М | М | Д | М | И | И | И | И | Д | Д |
| С | О | Н | Н | Е | | | | | С | Т |
| С | О | Н | | Е | Н | Е | А | Д | | |

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 (М – совпадение, ничего делать не надо; Р – заменить символ на другой; И – вставить символ на текущую позицию; Д – удалить символ из строки); вторая строка – исходная строка A; третья строка – исходная строка B.

Выполнение работы

Редакционное расстояние между двумя строками — это минимальное количество операций, необходимых для превращения одной строки в другую. Каждая операция имеет определенную цену, отражая разную вероятность разных ошибок при вводе текста, и т. п. Для решения задачи о редакционном расстоянии необходимо найти последовательность замен, минимизирующую суммарную цену. С помощью алгоритма Вагнера — Фишера создается матрица D размером $(n+1) \times (m+1)$, где: n — длина исходной строки A , m — длина целевой строки B . D — матрица для хранения расстояний между всеми префиксами первой строки и всеми префиксами второй строки, расстояние между двумя полными строками — последнее вычисленное значение.

Алгоритм Вагнера — Фишера с дополнительной операцией замены одного символа на два символа:

1. Значения матрицы заполняются "бесконечностью" (INF), кроме $D[0][0] = 0$ (преобразование пустой строки в пустую не требует операций).
2. Инициализация первой строки и столбца:
 - a. Первый столбец ($D[i][0]$): стоимость последовательного удаления всех символов из A : $i * \text{delete_cost}$
 - b. Первая строка ($D[0][j]$): стоимость последовательной вставки всех символов в B : $j * \text{insert_cost}$
3. Заполнение матрицы: для каждой пары индексов (i, j) (где $1 \leq i \leq n$, $1 \leq j \leq m$) вычисляется минимальная стоимость преобразования подстроки $A[0..i-1]$ в $B[0..j-1]$. Рассматриваются четыре возможные операции:
 - a. Удаление символа
Стоимость: $D[i-1][j] + \text{delete_cost}$.
 - b. Вставка символа
Стоимость: $D[i][j-1] + \text{insert_cost}$.
 - c. Совпадение или замена
Стоимость: $D[i-1][j-1]$, если $A[i-1] == B[j-1]$ (символы совпадают). $D[i-1][j-1] + \text{replace_cost}$, если требуется замена.

d. Удаление двух подряд идущих разных символов из A

Если $i \geq 2$ и $A[i-2] \neq A[i-1]$:

Стоимость: $D[i-2][j] + \text{delete_two_cost}$

Минимальное значение из этих четырех вариантов записывается в $D[i][j]$.

Итоговое редакционное расстояние находится в ячейке $D[n][m]$. Оно отражает минимальную стоимость преобразования всей строки A в строку B с учетом заданных стоимостей операций.

Редакционное предписание — последовательность действий, необходимых для получения из первой строки второй кратчайшим образом. Действия обозначаются так: D (англ. delete) — удалить, I (англ. insert) — вставить, R (англ. replace) — заменить, M (англ. match) — совпадение, [DT] (англ. delete two) — удаление двух разных символов подряд.

Алгоритм обратного отслеживания (backtracking) позволяет восстановить последовательность операций, которая привела к минимальной стоимости преобразования строки A в строку B. Этот процесс выполняется путем анализа матрицы расстояний, заполненной алгоритмом Вагнера-Фишера, и движения от правого нижнего угла матрицы ($D[n][m]$) к началу ($D[0][0]$).

Алгоритм обратного отслеживания:

1. Начинаем с позиции $i = \text{len}(A)$, $j = \text{len}(B)$ (правый нижний угол матрицы). На каждом шаге определяем, какая операция была применена для достижения текущей ячейки $D[i][j]$.

2. На каждом шаге проверяются возможные переходы в матрице:

a. Удаление двух разных символов (Delete two, DT)

Если $i \geq 2$, $A[i-2] \neq A[i-1]$, и $D[i][j] == D[i-2][j] + \text{delete_two_cost}$

Добавляется операция [D2], $i -= 2$

b. Вставка (Insert, I)

Проверяется условие: $D[i][j] == D[i][j-1] + \text{insert_cost}$. Вставка символа $B[j-1]$ в $A[i-1]$. Добавляется операция I, индекс j уменьшается: $j -= 1$.

с. Удаление (Delete, D)

Проверяется условие: $D[i][j] == D[i-1][j] + \text{delete_cost}$. Удаление символа $A[i-1]$. Добавляется операция D, индекс i уменьшается: $i -= 1$.

d. Совпадение (Match, M) или Замена (Replace, R)

Если ни одна из предыдущих проверок не сработала:

- Если $A[i-1] == B[j-1]$, добавляется операция M (совпадение).
- Иначе — операция R (замена $A[i-1]$ на $B[j-1]$).

Индексы i и j уменьшаются: $i -= 1, j -= 1$.

Операции записываются в обратном порядке (от конца к началу), поэтому после завершения цикла выполняется `reverse()`, чтобы получить правильную последовательность.

Оценка сложности

1. Построение матрицы расстояний

- Сложность по времени: $O(n \cdot m)$, где n и m — длины строк A и B . Каждая ячейка матрицы D размером $(n+1) \times (m+1)$ заполняется за константное время.
- Сложность по памяти: $O(n \cdot m)$. Хранится вся матрица расстояний.

2. Восстановление редакционного предписания

- Сложность по времени: $O(n + m)$. Алгоритм движется от $D[n][m]$ к $D[0][0]$, выполняя не более $n + m$ шагов. На каждом шаге проверяются условия за константное время.
- Сложность по памяти: $O(n + m)$. Хранится список операций длиной до $n + m$.

3. Общая оценка

- Сложность по времени: $O(n \cdot m)$
- Сложность по памяти: $O(n \cdot m)$

Построение матрицы определяет общую асимптотику, так как $O(n \cdot m) \gg O(n + m)$ для больших n и m .

Тестирование

Результаты тестирования программы представлены в табл. 1.

Табл. 1

| Входные данные | Выходные данные |
|------------------------------------|--|
| 1 1 1 2 entrance reenterable | IMIMMIMMRRM entrance reenterable |
| 1 1 1 2 reent nteent | RIMMMM reent nteent |
| 1 2 3 4 wooreeing woretnng | MMR[DT]MRMM wooreeing woretnng |

Исходный код программы см. в прил. А.

Выводы

В лабораторной работе был реализован алгоритм Вагнера-Фишера с дополнительной операцией. На основе матрицы расстояний, которую получает алгоритм, построен алгоритм нахождения редакционного предписания.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
INF = float('inf')

def read_input():
    replace_cost, insert_cost, delete_cost, delete_two_cost =
map(int, input().split())
    A = input().strip()
    B = input().strip()
    return replace_cost, insert_cost, delete_cost,
delete_two_cost, A, B

def initialize_dp(n, m, delete_cost, insert_cost):
    D = [[INF] * (m + 1) for _ in range(n + 1)]
    D[0][0] = 0
    for i in range(1, n + 1):
        D[i][0] = D[i-1][0] + delete_cost
    for j in range(1, m + 1):
        D[0][j] = D[0][j-1] + insert_cost
    return D

def fill_dp(D, A, B, replace_cost, insert_cost, delete_cost,
delete_two_cost):
    n = len(A)
    m = len(B)
    for i in range(n + 1):
        for j in range(m + 1):
            if i > 0:
                D[i][j] = min(D[i][j], D[i-1][j] + delete_cost)
            if j > 0:
                D[i][j] = min(D[i][j], D[i][j-1] + insert_cost)
            if i > 0 and j > 0:
                cost = 0 if A[i-1] == B[j-1] else replace_cost
                D[i][j] = min(D[i][j], D[i-1][j-1] + cost)
            if i >= 2 and A[i-1] != A[i-2]:
                D[i][j] = min(D[i][j], D[i-2][j] +
delete_two_cost)

def backtrack_operations(D, A, B, replace_cost, insert_cost,
delete_cost, delete_two_cost):
    operations = []
    i = len(A)
    j = len(B)

    print("\nПошаговое восстановление операций:")
    print("(Ищем путь от правого нижнего угла к началу матрицы)")

    while i > 0 or j > 0:
        print(f"\nПозиция: A[{i}]='{A[i-1] if i>0 else ' '}',
B[{j}]='{B[j-1] if j>0 else ' '}'")
        print(f"Текущая стоимость: {D[i][j]}")

        if i >= 2 and D[i][j] == D[i-2][j] + delete_two_cost and
A[i-1] != A[i-2]:
```

```

        print(f"[DT]: удаление двух разных символов '{A[i-2]}'
{A[i-1]]}'")
        operations.append('[DT]')
        i -= 2
    elif j > 0 and D[i][j] == D[i][j-1] + insert_cost:
        print(f"I: вставка '{B[j-1]]}'")
        operations.append('I')
        j -= 1
    elif i > 0 and D[i][j] == D[i-1][j] + delete_cost:
        print(f"D: удаление '{A[i-1]]}'")
        operations.append('D')
        i -= 1
    else:
        if i > 0 and j > 0 and A[i-1] == B[j-1]:
            print(f"M: совпадение '{A[i-1]]}'")
            operations.append('M')
        else:
            print(f"R: замена '{A[i-1]]' на '{B[j-1]]}'")
            operations.append('R')
        i -= 1
        j -= 1

    print(f"Текущие операции: {list(reversed(operations))}")
    print()
    operations.reverse()
    return operations

def print_dp_matrix(D, A, B):
    print("\nМатрица минимальных стоимостей:")

    n = len(A)
    m = len(B)
    col_width = 4

    header = " " * 8 + "".join([f"{char:^{col_width}}" for char
in B])
    print(header)

    for i in range(n + 1):
        row_label = ' ' if i == 0 else A[i-1]
        row = []
        for j in range(m + 1):
            val = D[i][j]
            row.append(" ∞ " if val == INF else f"{val:3d}")
        row_str = " ".join([f"{item:^{col_width-1}}" for item in
row])
        print(f"{row_label:2} {row_str}")

def print_result(operations, A, B):
    print("\nРезультат:")
    print(''.join(operations))
    print(A)
    print(B)

def main():
    replace_cost, insert_cost, delete_cost, delete_two_cost, A, B
= read_input()
    n = len(A)
    m = len(B)

```

```

        D = initialize_dp(n, m, delete_cost, insert_cost)
        fill_dp(D, A, B, replace_cost, insert_cost, delete_cost,
delete_two_cost)

        print_dp_matrix(D, A, B)
        operations = backtrack_operations(D, A, B, replace_cost,
insert_cost, delete_cost, delete_two_cost)
        print_result(operations, A, B)

if __name__ == '__main__':
    main()

```