

МІНІСТЕРСТВО НАУКИ ТА ОСВІТИ УКРАЇНИ

Київський національний університет імені Тараса Шевченка

Факультет комп'ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Лабораторна робота №2

Тема: «Алгоритми на матрицях.»

З дисципліни «Алгоритми та складність»

Варіант №7 - Обчислити відстань Левенштейна

Виконала:

студентка групи ІПС - 21

Яцкова Ія Григорівна

Київ 2025

## Зміст

1. Постановка задачі .....	3
2. Теоретична частина .....	4
3. Алгоритм .....	5
4. Алгоритм функції покрокового виводу .....	6
5. Складність алгоритму.....	7
6. Інтерфейс користувача.....	8
7. Тест програми.....	9
8. Тест для перевірки часу роботи програми.....	10

## **1. Постановка задачі**

**Тема:** Алгоритми на матрицях.

**Завдання:**

1. Обчислити відстань Левенштейна.
2. Реалізувати алгоритм відповідно до варіанту.
3. Вивести на екран послідовність дій для перетворення первого рядка в другий.

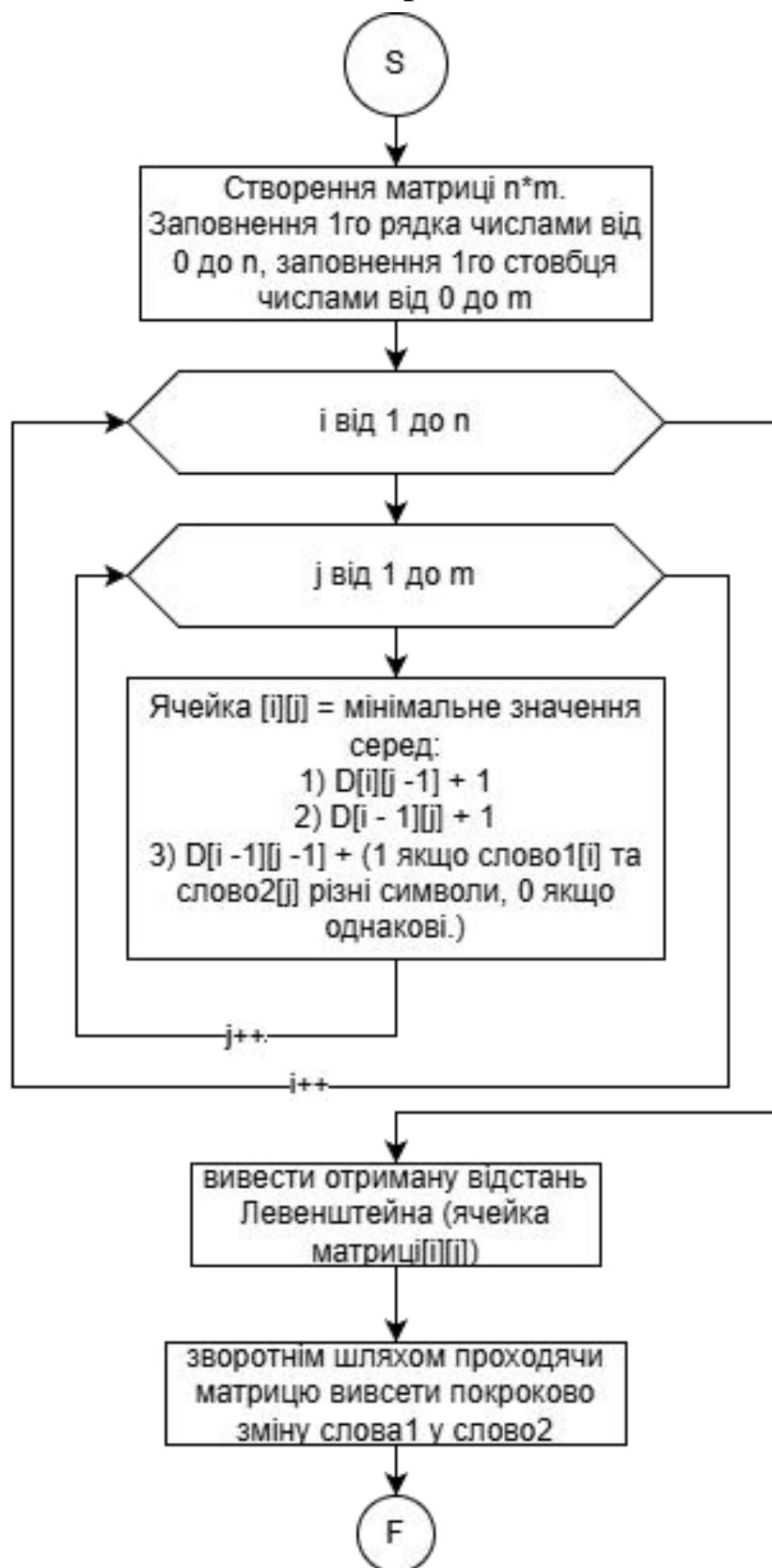
## 2. Теоретична частина

- Відстань Левенштейна - метрика подібності між двома строковими послідовностями. Чим більша відстань, тим більше відрізняються рядки. Для двох одинакових послідовностей відстань = 0. По суті, це мінімальне число односимвольних перетворень, необхідних для того, щоб перетворити одну послідовність на іншу.
- Алгоритм Левенштейна використовується для визначення мінімальної кількості операцій, необхідних для перетворення одного рядка в інше.
- До дозволених операцій належать:
  - вставка символу;
  - видалення символу;
  - заміна одного символу іншим;
- Для обрахунку відстані потрібно створити матрицю  $m \times n$ , де  $m$  - довжина 1 слова,  $n$  - довжина 2го,  $S1[i]$  - i-тий символ слова 1,  $S2[j]$  - j-тий символ слова 2.  
 $m(S1[i], S2[j]) = 1$ , якщо символи  $S1[i]$  та  $S2[j]$  **не дорівнюють** один одному,  
 $m(S1[i], S2[j]) = 0$ , якщо символи  $S1[i]$  та  $S2[j]$  **дорівнюють** один одному.  
Матриця заповнюється таким чином:

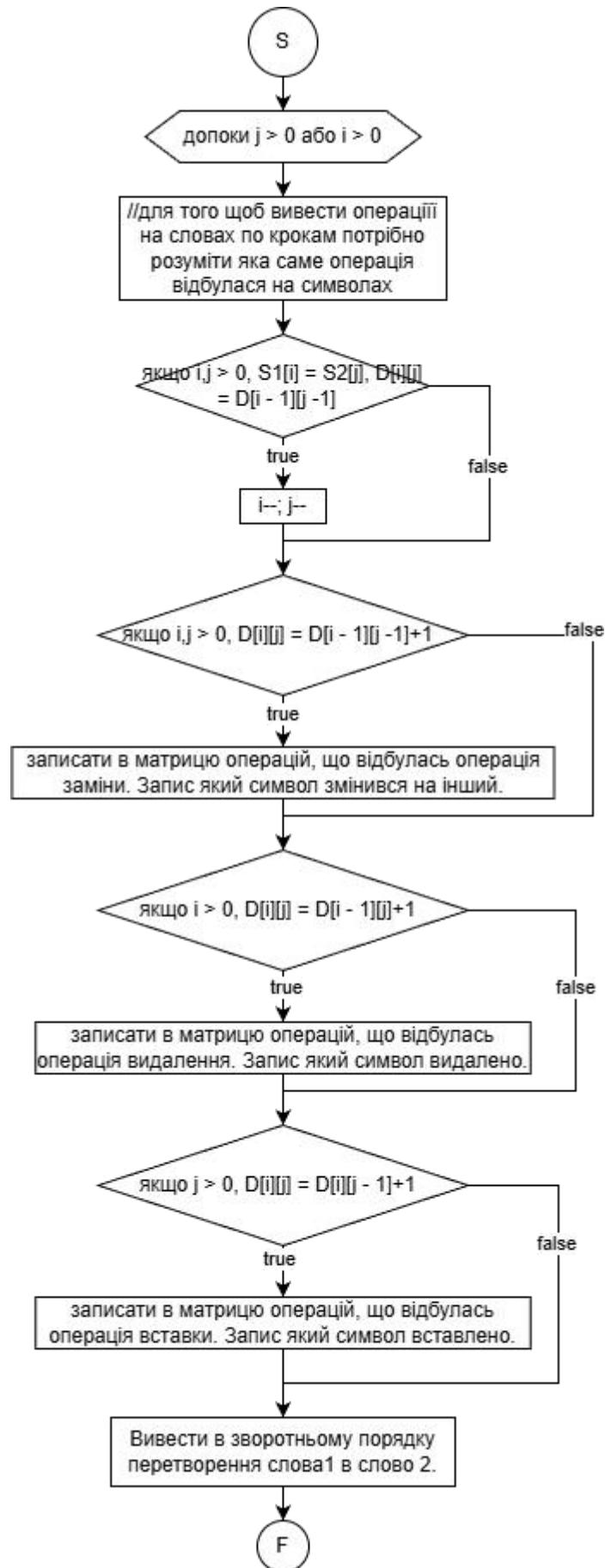
$$D(i, j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & j = 0, i > 0 \\ j, & i = 0, j > 0 \\ \min\{ & \\ & D(i, j - 1) + 1, \\ & D(i - 1, j) + 1, \\ & D(i - 1, j - 1) + m(S1[i], S2[j]) \} \end{cases}$$

Таким чином у ячейці  $D(m; n)$  буде відстань Левенштейна між словом 1 і словом 2.

### 3. Алгоритм



#### 4. Алгоритм функції покрокового виводу



## **5. Складність алгоритму**

$T(n) = O(n*m)$

### **Мова реалізації алгоритму**

C++

### **Модулі програми**

**1) int levenshtein\_index(string word1, string word2);**

Метод знаходження відстані Левенштейна.

**2) int size\_of\_str(string word);**

Метод знаходження реальної довжини строки без /0 символа.

**3) int equal(char a, char b);**

Метод порівнює символи, якщо вони однакові повертає 1, якщо різні 0.

**4) int trio\_min(int a, int b, int c);**

Метод знаходить найменше серед трьох чисел. Повертає найменше.

**5) vector<Op> backtrace\_ops(const string& word1, const string& word2, const vector<vector<int>>& matrix);**

Метод запису всіх операцій, які перетворюють слово1 в слово2.

**6) void show\_step\_by\_step(string word1, const string& word2, const vector<Op>& ops);**

Метод виводу на екран покрокового перетворення слова1 в слово2.

## **6.Інтерфейс користувача**

Введення даних відбувається через консоль. Спочатку на екран виводиться перелік пронумерованих операцій, які користувач може виконати.

Вхідні дані: Номер операції яка вас цікавить. Доступні такі операції як:

- 1) Розрахувати відстань Левенштейна.
- 0) Вихід з програми.

## 7. Тест програми

Нехай є два слова brat та cat. Розрахуємо вручну відстань Левенштейна і за допомогою програми.

1) Побудуємо матрицю і заповнимо її коефіцієнтами.

$i$	$j$	B	R	A	T
1 <sup>o</sup> ) $D(1; 1)$ :		0	1	2	3
$\min(1+1; 2; 1) = 1$	C	1	1	2	3
2 <sup>o</sup> ) $D(2; 1)$ :		A	2	2	2
$\min(3; 2; 2) = 2$	T	3	3	3	2
3 <sup>o</sup> ) $D(1; 1)$ : $\min(2; 2; 1) = 1$					
2 <sup>o</sup> ) $D(2; 1)$ : $\min(3; 2; 2) = 2$					
3 <sup>o</sup> ) $D(3; 1)$ : $\min(4; 3; 3) = 3$					
4 <sup>o</sup> ) $D(4; 1)$ : $\min(5; 4; 4) = 4$					
5 <sup>o</sup> ) $D(1; 2)$ : $\min(2; 3; 2) = 2$					
6 <sup>o</sup> ) $D(2; 2)$ : $\min(3; 3; 2) = 2$					
7 <sup>o</sup> ) $D(3; 2)$ : $\min(4; 3; 2) = 2$					
8 <sup>o</sup> ) $D(4; 2)$ : $\min(5; 3; 4) = 3$					
9 <sup>o</sup> ) $D(1; 3)$ : $\min(3; 4; 3) = 3$					
10 <sup>o</sup> ) $D(2; 3)$ : $\min(3; 4; 3) = 3$					
11 <sup>o</sup> ) $D(3; 3)$ : $\min(3; 4; 3) = 3$					
12 <sup>o</sup> ) $D(4; 3)$ : $\min(4; 4; 2) = 2$					

Вручну:

```

index (1; 1) :1
index (1; 2) :2
index (1; 3) :3
index (2; 1) :2
index (2; 2) :2
index (2; 3) :3
index (3; 1) :3
index (3; 2) :2
index (3; 3) :3
index (4; 1) :4
index (4; 2) :3
index (4; 3) :2

```

За допомогою програми:

Проаналізувавши матрицю і знаючи відстань виводимо операції:

```

==== Step by step transformation ====
Before: brat
Operation: DELETE 'b' at pos 0
After: rat
Target: cat
-----
Before: rat
Operation: SUBSTITUTE 'r' -> 'c' at pos 0
After: cat
Target: cat
-----
Final result: cat -> cat
final index: 2

```

Покроковий запис показує, що алгоритм працює коректно.

## 8. Тест для перевірки часу роботи програми

Для матриці розміру  $8 \times 5$  заміряється час роботи функцій розрахунку відстані Левенштейна і покрокового виводу змін слова 1 у слово 2.

за допомогою бібліотеки **<chrono>**:

```
auto start = std::chrono::high_resolution_clock::now();
void levenshtein_realisation(string word1, string word2);
auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = end - start;
std::cout << "Time taken: " << duration.count() << " seconds" << std::endl;
```

Якщо обчислення виконуються без помилки,  
а час не перевищує встановлений ліміт (наприклад, 0.05 сек для  $8 \times 5$ ),  
тест вважається **PASSED**.

```
Time taken: 0.038999 seconds
```

Результат роботи коректний.

## **9. Висновок**

У даній роботі було розглянуто алгоритм обчислення відстані Левенштейна, а також принципи побудови таблиці динамічного програмування та методи відновлення послідовності операцій перетворення одного слова в інше. Алгоритм Левенштейна визначає мінімальну кількість елементарних операцій — вставки, видалення та заміни символів — необхідних для перетворення одного рядка в інший.

Алгоритм базується на побудові матриці розміром  $(n+1) \times (m+1)$ , де  $n$  і  $m$  — довжини вхідних слів. Кожен елемент таблиці обчислюється на основі трьох попередніх значень, що відповідають можливим операціям редактування. Заповнивши таблицю згідно з рекурентною формулою, у правому нижньому елементі отримуємо значення мінімальної «вартості» перетворення. Для відновлення конкретної послідовності операцій застосовується зворотний прохід по матриці, що дозволяє покроково визначити, які редактування були виконані.

Завдяки такому підходу алгоритм Левенштейна є ефективним інструментом для задач пошуку подібності рядків, автоматичного виправлення помилок, порівняння текстів та задач обробки природної мови. Його використання дозволяє точно оцінити ступінь відмінності між словами та забезпечує формальний спосіб побудови оптимального перетворення одного рядка в інший.

## **10. Джерела інформації**

1. Алгоритми і структура даних: Навчальний посібник / В.М.Ткачук. - Івано-Франківськ : Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016.-286 с.
2. <https://habr.com/eng/articles/676858/>