**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №4

«Модульное тестирование»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Сомов Кирилл |  | Гапанюк Ю. Е. |
| Подпись и дата: |  | Подпись и дата: |
|  |  |  |

Москва, 2023 г.

Задание

Написать тесты для одной из прошлых ЛР.

Код программы

Main.py

from levenstein\_distance import LevenshteinDistance  
  
  
def main():  
 word1 = input("Введите первое слово: ")  
 word2 = input("Введите второе слово: ")  
  
 distance = LevenshteinDistance.levenshtein\_distance(word1, word2)  
 print(f"Расстояние Левенштейна между '{word1}' и '{word2}': {distance}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

levenstein\_distance.py

class LevenshteinDistance:  
 @staticmethod  
 def levenshtein\_distance(word1, word2):  
 m, n = len(word1), len(word2)  
 # Создаем матрицу размером (m+1) x (n+1)  
 dp = [[0] \* (n + 1) for \_ in range(m + 1)]  
  
 # Инициализируем первую строку и первый столбец  
 for i in range(m + 1):  
 dp[i][0] = i  
 for j in range(n + 1):  
 dp[0][j] = j  
  
 # Заполняем матрицу  
 for i in range(1, m + 1):  
 for j in range(1, n + 1):  
 if word1[i - 1] != word2[j - 1]:  
 dp[i][j] = min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]) + 1  
 else:  
 dp[i][j] = dp[i - 1][j - 1]  
 return dp[m][n]

test\_levenshtein\_tdd.py

import unittest  
from levenstein\_distance import LevenshteinDistance  
  
  
class TestLevenshteinDistance(unittest.TestCase):  
 def test\_same\_words(self):  
 self.assertEqual(LevenshteinDistance.levenshtein\_distance("abc", "abc"), 0)  
  
 def test\_different\_words\_different\_length(self):  
 self.assertEqual(LevenshteinDistance.levenshtein\_distance("kitten", "sitting"), 3)  
  
 def test\_different\_words\_same\_length(self):  
 self.assertEqual(LevenshteinDistance.levenshtein\_distance("flaw", "lawn"), 2)  
  
 def test\_null\_words(self):  
 self.assertEqual(LevenshteinDistance.levenshtein\_distance("", ""), 0)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

test\_levenshtein\_bdd.py

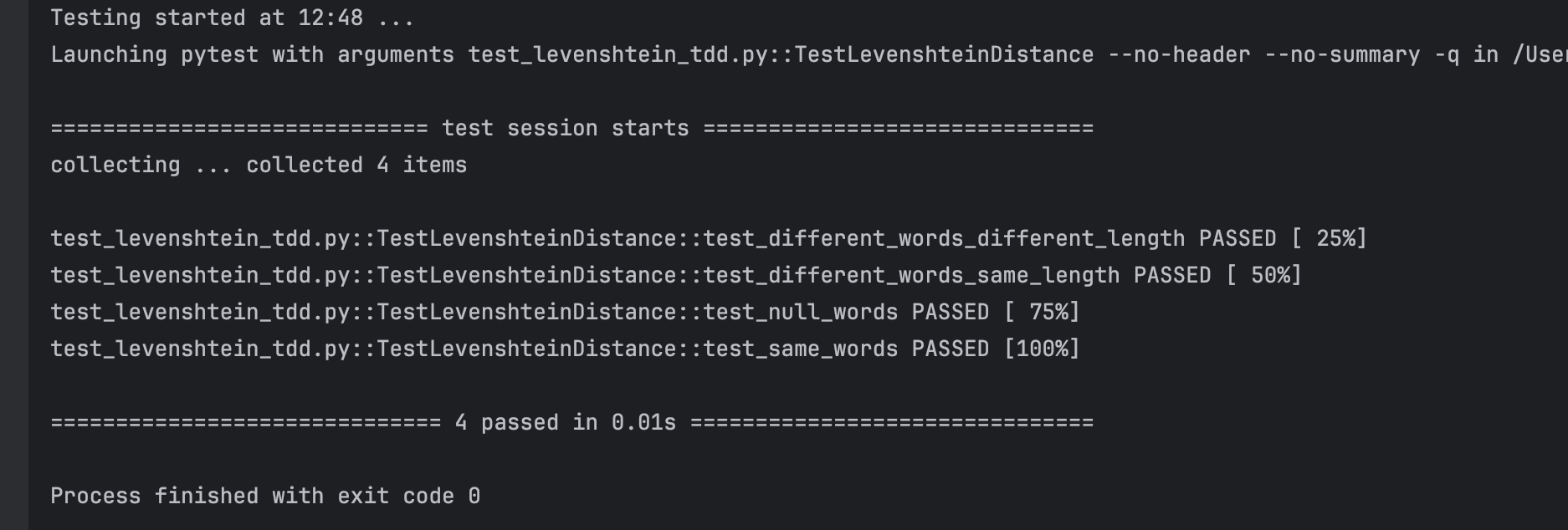
# test\_levenshtein\_bdd.py  
import pytest  
from levenstein\_distance import LevenshteinDistance  
  
  
@pytest.mark.parametrize("word1, word2, expected\_distance", [  
 ("abc", "abd", 1),  
 ("abcd", "abc", 1),  
 ("abc", "abc", 0),  
 ("", "", 0),  
 ("kitten", "sitting", 3),  
 ("flaw", "lawn", 2),  
 ("developer", "develop", 2),  
])  
def test\_levenshtein\_distance(word1, word2, expected\_distance):  
 # When  
 distance = LevenshteinDistance.levenshtein\_distance(word1, word2)  
 # Then  
 return distance == expected\_distance

Levenshtein\_distance.feature

Feature: Levenshtein Distance Calculation  
  
 Scenario Outline: Calculate Levenshtein Distance  
 Given I have two words <word1> and <word2>  
 When I calculate the Levenshtein distance between the words  
 Then the distance should be <expected\_distance>  
  
 Examples:  
 | word1 | word2 | expected\_distance |  
 | abc | abc | 0 |  
 | abc | abcd | 1 |  
 | abc | abd | 1 |  
 | | | 0 |  
 | kitten | sitting | 3 |  
 | flaw | lawn | 2 |  
 | developer | develop | 3 |

Вывод

tdd



Bdd

