**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**

Вариант 7a.

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Трунов Б.Г. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2025

**Цель работы:**

Изучить алгоритмы Левенштейна для нахождения редукционного расстояния. Также реализовать задание по варианту.

**Задание.**

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 -> 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

**Реализация**

**Описание алгоритма Левенштейна:**

Программа реализует алгоритм Левенштейна, который является классическим методом динамического программирования для вычисления редакционного расстояния между двумя строками. Редакционное расстояние — это минимальное количество операций редактирования (вставка, удаление, замена), необходимых для преобразования строки *S* в строку *T*.

**Шаги алгоритма**

* Инициализация таблиц
  + Таблица *dp* размером *(m+1)\*(n+1)*, где *m* – длина строки *s*, *n* – длина строки *t*. Эта таблица хранит минимальную стоимость преобразований первых *i* символов *s* в первые *j* символов *t*.
  + Таблица ops размером *(m+1)\*(n+1)*, где *m* – длина строки *s*, *n* – длина строки *t*. Эта таблица хранит операции (*‘M’* - совпадение, *‘R’* – замена, *‘I’* – вставка, *‘D’* – удаление) для каждой ячейки, для восстановления шагов.
  + Заполняем первый столбец *dp[i][0]* и строку *dp[0][j]*
    - Для *i=0…m*: *dp[i][0]=i\*cost\_del*(стоимость удаления *i* символов из *s*), в *ops[i][0]* ставим *‘D’*(удаление), кроме *ops[0][0]=''*.
    - Для *j=0...n*: *dp[0][j]=j\*cost\_ins*(стоимость вставки *j* символов из *T*), в *ops[0][j]* ставим *‘I’*(вставка), кроме *ops[0][0]= ''*.
* Заполнение таблиц
  + Для каждой ячейки *(i,j)* где *(i=1…m, j=1…n)*:
    - Если символы совпадают *(S[i-1]=T[j-1])*:
      * Стоимость не увеличивается: *dp[i][j]=dp[i-1][j-1]*
      * Записываем в *ops[i][j]= 'M'*
    - Если символы различные:
      * Считаем три возможных стоимости:
        + Вставка: *dp[i][j-1] + cost\_ins*(добавляем символ *T[j-1]*)
        + Удаление: *dp[i-1][j] + cost\_del*(удаляем символ *S[i-1]*)
        + Замена: *dp[i-1][j-1] + cost\_repl*(меняем *S[i-1]* на *T[j-1]*)
      * Выбираем минимальную стоимость и записываем её в *dp[i][j]*
      * В *ops[i][j]* записываем операцию, давшую минимальную стоимость.
    - После заполнения таблицы каждая ячейка будет означать как дешево дойти до текущего состояния и какая операция необходима.
* Сбор шагов (восстановление последовательности операций):
  + Начинаем с ячейки *(m,n)* – это конец преобразования(вся строка *S* превращается в *T*).
  + Создаём пустой список для операций.
  + Пока не дойдём до *(0,0)*, смотрим на *ops[i][j]*
    - Если *‘M’* или *‘R’*:
      * Добавляем *‘M’* или *‘R’* в список
      * Переходим к *(i-1,j-1)*, так как обработали символы из обеих строк.
    - Если *‘I’*
      * Добавляем *‘I’* в список
      * Переходим к *(i,j-1)*, так как добавили символ из *T*, но не трогали *S*.
    - Если *‘D’*
      * Добавляем *‘D’* в список
      * Переходим к *(i-1,j)*, так как удалили символ из *S*, но не трогали *T*.
  + Разворачиваем список.

**Описание функций и структур:**

* *dp: List[List[int/float]]:* Таблица для хранения минимальных стоимостей редактирования. В *classic\_levenshtein\_distance* использует *int*, в *cursed\_levenshtein\_distance — float* (для обработки *float('inf')* при запрещённых операциях).
* *ops: List[List[str]]:* Таблица для хранения операций (*'M'* — совпадение, *'R'* — замена, *'I'* — вставка, *'D'* — удаление, *'X'* — отсутствие допустимых операций).
* *cursed\_set: Set[int]:* Множество "проклятых" индексов, где ограничены операции (удаление/замена).
* *operations: List[str]:* Список операций, восстанавливающий последовательность редактирования.
* *classic\_levenshtein\_distance(s: str, t: str, w\_ins: int, w\_del: int, w\_sub: int) -> Tuple[int, List[str]]* Вычисляет классическое расстояние Левенштейна между строками *s* и *t* с весами операций: вставка (*cost\_ins*), удаление (*cost\_del*), замена (*cost\_repl*).
* *restrict\_operations(index: int, char: str, cursed\_set: Set[int]) -> Tuple[bool, bool]* Определяет допустимость операций удаления и замены для символа по индексу *index*.
* *cursed\_levenshtein\_distance(s: str, t: str, cursed\_indices: List[int], w\_ins: int, w\_del: int, w\_sub: int) -> Tuple[int, List[str]]* Вычисляет расстояние Левенштейна с ограничениями на операции для "проклятых" индексов.

**Оценка сложности алгоритма:**

**Временная сложность:**

* Создание и заполнение таблиц *O(m\*n)*, где *m* длина строки *s*, *n* – длина строки *t*.
* Восстановление операций: проход по таблице *ops* от *(m,n)* до *(0,0)*, не более *O(m+n)*

**Итог:** *O(n⋅m)*

**Пространственная сложность**

* Хранение таблиц *dp* и *ops* *O(m\*n)*
* Хранение списка шагов *O(m+n)*

Итог: *O(n⋅m)*

**Тестирование**

Таблица 1. Тестирование.

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| entrance  reenterable | 7 |
| cat  cat | 0 |
| cat  cot | 1 |
| dog  doing | 2 |
| helloasd | 8 |
| kitten  fishing | 5 |

**Вывод**

В ходе лабораторной работы были написаны программы с использованием алгоритма Левенштейна.