**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: Расстояние Левенштейна. Вариант 4а.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Поддубный В.А. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2025

## Цель работы

Нахождения редакционного предписания алгоритмом Вагнера-Фишера.

**Выполнение работы**

## Описание реализованного алгоритма

В данной работе реализован алгоритм вычисления редакционного расстояния Левенштейна с произвольными весами операций:  
- замены одного символа на другой (replace)  
- вставки символа (insert)  
- удаления символа (delete)  
- замены одного символа на два (replace-to-two)  
  
В коде используются две основные функции:  
- getLevenshteinDistance — вычисляет стоимость преобразования строк с учетом заданных операций.  
- getPrescription — восстанавливает последовательность операций, необходимых для получения строки B из строки A с минимальной стоимостью.

## Алгоритм расчета минимальной стоимости (getLevenshteinDistance)

Алгоритм строит матрицу dp размером (len(s1)+1) × (len(s2)+1), где dp[i][j] — минимальная стоимость преобразования первых i символов строки s1 в первые j символов строки s2.  
  
Инициализация:  
- Первая строка (dp[0][j]) заполняется стоимостью вставок.  
- Первый столбец (dp[i][0]) — стоимостью удалений.  
  
Заполнение матрицы:  
Для каждой позиции dp[i][j] вычисляется минимум из:  
- Удаление: dp[i-1][j] + deleteCost  
- Вставка: dp[i][j-1] + insertCost  
- Замена: 0, если символы равны, иначе replaceCost  
- Замена одного символа на два (если j ≥ 2): dp[i-1][j-2] + replaceToTwoCost

## Алгоритм восстановления последовательности операций (getPrescription)

После построения матрицы dp, функция getPrescription по обратному пути из dp[s1.length][s2.length] к dp[0][0] восстанавливает редакционное предписание.  
  
Для каждой позиции:  
- M — совпадение символов  
- R — замена  
- D — удаление  
- I — вставка  
- T — замена одного символа на два  
  
Формируется строка операций в обратном порядке, затем переворачивается.

## Оценка сложности алгоритма

По времени:  
Основной этап — заполнение матрицы размером (m+1) × (n+1), где m = len(s1), n = len(s2)  
Время: O(m⋅n)  
  
По памяти:  
Память: O(m⋅n) — требуется полная матрица для хранения промежуточных значений.

## Тестирование

Результаты тестирования представлены в таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1. | 1 1 1 1  qwerty  ytrewq | Редакционное расстояние: 5 | Алгоритм Вагнера-Фишера. Результат вычислен верно. |
| 2. | 5 2 3 1  aabc  lksa | Редакционное расстояние: 8 | Алгоритм Вагнера-Фишера. Результат вычислен верно. |
| 3. | 100 100 100 100  asdf  asdf | Редакционное расстояние: 0 | Алгоритм Вагнера-Фишера. Результат вычислен верно. |
| 4. | 1 1 1 1  wierghwij  sooidfhgi | 7  TRRRRMRMD  wierghwij  sooidfhgi | Алгоритм Вагнера-Фишера c восстановлением действий. Результат вычислен верно. |

Табл. 1. – Результаты тестирования

## Выводы

Был реализован алгоритм Вагнера-Фишера для вычисления редакционного предписания, определяя минимальное количество операций (вставки, удаления, замены) для преобразования одной строки в другую.

# Приложение А

package ru.rectid

import kotlin.math.min

const val DEBUG = true

const val PRINT\_DISTANCE = true

object Logger {

fun log(message: String) {

if (DEBUG) {

println(message)

}

}

}

fun main() {

val prices = readln().split(" ").map(String::toInt)

val insertCost = prices[1]

val deleteCost = prices[2]

val replaceCost = prices[0]

val replaceToTwoCost = prices[3]

val s1 = readln()

val s2 = readln()

val dp = getLevenshteinDistance(s1, s2, insertCost, deleteCost, replaceCost, replaceToTwoCost)

Logger.log("Расстояние Левенштейна: ${dp[s1.length][s2.length]}")

if (PRINT\_DISTANCE) {

println(dp[s1.length][s2.length])

}

println(getPrescription(s1, s2, dp, insertCost, deleteCost, replaceCost, replaceToTwoCost))

println(s1)

println(s2)

}

fun getLevenshteinDistance(s1: String, s2: String, insertCost: Int, deleteCost: Int, replaceCost: Int, replaceToTwoCost: Int): Array<IntArray> {

val dp = Array(s1.length + 1) { IntArray(s2.length + 1) { 0 } }

for (i in 1..s2.length) {

dp[0][i] = dp[0][i - 1] + insertCost

Logger.log("dp[0][$i] = ${dp[0][i]} (вставка $insertCost)")

}

for (i in 1..s1.length) {

dp[i][0] = dp[i - 1][0] + deleteCost

Logger.log("dp[$i][0] = ${dp[i][0]} (удаление $deleteCost)")

}

for (i in 1..s1.length) {

for (j in 1..s2.length) {

val costReplace = if (s1[i - 1] == s2[j - 1]) 0 else replaceCost

dp[i][j] = min(

dp[i - 1][j] + deleteCost,

min(

dp[i][j - 1] + insertCost,

dp[i - 1][j - 1] + costReplace

)

)

if (j >= 2) {

dp[i][j] = min(dp[i][j], dp[i - 1][j - 2] + replaceToTwoCost)

}

Logger.log("dp[$i][$j] = ${dp[i][j]} (замена ${if (costReplace == 0) "нет" else replaceCost}, вставка $insertCost, удаление $deleteCost, замена на два $replaceToTwoCost)")

}

}

return dp

}

fun getPrescription(s1: String, s2: String, dp: Array<IntArray>, insertCost: Int, deleteCost: Int, replaceCost: Int, replaceToTwoCost: Int): String {

val sb = StringBuilder()

var i = s1.length

var j = s2.length

while (i > 0 || j > 0) {

when {

i > 0 && j > 0 && s1[i - 1] == s2[j - 1] -> {

sb.append("M")

Logger.log("M (совпадение: s1[${i - 1}] = s2[${j - 1}])")

i--

j--

}

i > 0 && j > 0 && dp[i][j] == dp[i - 1][j - 1] + replaceCost -> {

sb.append("R")

Logger.log("R (замена: s1[${i - 1}] = s2[${j - 1}])")

i--

j--

}

i > 0 && dp[i][j] == dp[i - 1][j] + deleteCost -> {

sb.append("D")

Logger.log("D (удаление: s1[${i - 1}])")

i--

}

j > 0 && dp[i][j] == dp[i][j - 1] + insertCost -> {

sb.append("I")

Logger.log("I (вставка: s2[${j - 1}])")

j--

}

i > 0 && j >= 2 && dp[i][j] == dp[i - 1][j - 2] + replaceToTwoCost -> {

sb.append("T")

Logger.log("T (замена одного символа на два: s1[${i - 1}] на два символа в s2)")

i--

j -= 2

}

}

}

return sb.reverse().toString()

}