Государственное образовательное учреждение высшего профессионального образования
«Московский Государственный Технический Университет им Н. Э. Баумана.»

Отчет
По лабораторной работе №1
По курсу «Анализ Алгоритмов»

Тема: «Редакционное расстояние Левенштейна»

Студент: Губайдуллин Р.Т.
Группа: ИУ7-51

Преподаватели: Волкова Л.Л.
Строганов Ю.В.

Постановка задачи

В ходе лабораторной работы необходимо:
1. Изучить алгоритм Левенштейна
2. Реализовать следующие алгоритмы:
    1. Алгоритм Левенштейна с использованием рекурсии
    2. Алгоритм Левенштейна с использованием матрицы
    3. Модифицированный алгоритм Левенштейна с использованием матрицы
3. Сравнить базовый и рекурсивный алгоритмы

Описание алгоритма

Алгоритм Левенштейна - алгоритм поиска минимального редакционного расстояния между двумя строками. Результатом работы алгоритма является минимальное количество операций вставки, удаления и замены одного символа на другой, необходимые для превращения одной строки в другую.

Допустимые операции:
- Замена символа (R - replace)
- Вставка символа (I - insert)
- Удаление символа (D - delete)
- Совпадение символов (M - match)

Операции замены, вставки и удаления имеют цену 1, совпадение - 0.
Рассчитать редакционное расстояние можно по рекуррентной формуле:

$$D(i,j) = \begin{cases} 0 & if\ i = 0,\ j = 0 \\ i & if\ i > 0\ j = 0 \\ j & if\ i = 0\ j > 0 \\ min(D(i,j-1)+1,\ D(i-1,j),\ D(i-1,j-1)+m(S_1[i],S_2[j])) & if i > 0,\ j > 0 \end{cases}$$

Где $m(a,b) = 0$, если $a = b$, 1 - иначе; min - возвращает наименьший из аргументов.

Например, для строк CONNECT и CONEHEAD можно построить следующую таблицу преобразований:

M	**M**	**M**	**R**	**I**	**M**	**R**	**R**
C	O	N	N		E	C	T
C	O	N	E	H	E	A	D

Использование матрицы для расчета редакционного расстояния:
Если длины строк str1 и str2 равны M и N соответственно, то найти расстояние Левенштейна можно используя матрицу размерностью $(M+1) * (N+1)$.

Пример для строк POLYNOMIAL и EXPONENTIAL:

		P	**O**	**L**	**Y**	**N**	**O**	**M**	**I**	**A**	**L**
	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9

N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	7
L	11	10	9	8	9	8	8	8	8	7	6

Таблица заполняется с ячейки (0,0)

В каждой ячейке содержится количество операций над частью первой подстроки, чтобы преобразовать её в часть второй подстроки.

Модифицированный алгоритм имеет дополнительную операцию: перестановка двух соседних символов. Цена - 1.

Реализация алгоритмов

Листинг: рекурсивная форма алгоритма.

```
int LevenshteinRec(std::string s1, int n, std::string s2, int m, int& count) {
	if (n == 0 && m == 0)
		return 0;
	if (n == 0)
		return m;
	if (m == 0)
		return n;

	int insert = LevenshteinRec(s1, n, s2, m - 1, ++count) + 1;
	int del = LevenshteinRec(s1, n - 1, s2, m, ++count) + 1;
	int repl = LevenshteinRec(s1, n - 1, s2, m - 1, ++count) + (s1[n - 1] == s2[m - 1] ? 0 : 1);



	return CustomMin(repl, del, insert);
}
```

Входные данные: str1 - первая строка, n - длина первой строки, str2- вторая строка, m - длина второй строки.

Выходные данные: Значение редакционного расстояния.

Листинг: Алгоритм Левенштейна с использованием матрицы

```
void LevenshteinMatrix(std::string str1, std::string str2, Matrix &matrix) {
    int n = str1.size();
    int m = str2.size();

    //…

    int cost = 0;

    for (int i = 0; i < n + 1; ++i)
        matrix.m[i][0] = i;
    for (int i = 0; i < m + 1; ++i)
        matrix.m[0][i] = i;

    for (int i = 1; i < n + 1; ++i) {
        for (int j = 1; j < m + 1; ++j) {
            cost = (str1[i - 1] == str2[j - 1] ? 0 : 1);
            matrix.m[i][j] = CustomMin(matrix.m[i - 1][j] + 1,
                                       matrix.m[i][j-1] + 1,
                                       matrix.m[i-1][j-1] + cost);
        }
    }

    //…
    return;
}
```

Листинг: Модифицированный алгоритм Левенштейна с использованием матрицы

```
void LevenshteinModifMatrix(std::string str1, std::string str2, Matrix &matrix) {
    int n = str1.size();
    int m = str2.size();

    //…
    for (int i = 0; i < n + 1; ++i)
        matrix.m[i][0] = i;
    for (int i = 0; i < m + 1; ++i)
        matrix.m[0][i] = i;


    for (int i = 1; i < n + 1; ++i) {
        for (int j = 1; j < m + 1; ++j) {
            int insert = matrix.m[i - 1][j] + 1;
            int del = matrix.m[i][j - 1] + 1;
            int repl = matrix.m[i - 1][j - 1] + (str1[i - 1] == str2[j - 1] ? 0 : 1);

            int min = CustomMin(insert, del, repl);

            if (i > 1 && j > 1 && str1[i - 1] == str2[j - 2] && str1[i - 2] == str2[j - 1]) {
                int swap = matrix.m[i - 2][j - 2] + 1;
                min = (swap < min ? swap : min);
            }
            matrix.m[i][j] = min;
        }
    }
    //…

    return;
}
```

Тесты

Слово-1	Слово-2	Базовый алгоритм	Модифицированный алгоритм	Операции
	Cat	3	3	Пустая строка (добавление)
Cat		3	3	Пустая строка (удаление)
cat	cap	1	1	Замена
wood	woodo	1	1	Добавление
woodo	wood	1	1	Удаление
wood	wodo	2	1	Перестановка
wood	wood	0	0	Одинаковые строки

Заключение

Алгоритм Левенштейна позволяет решать множество задач: автоматическое исправление ошибок в слове, сравнение файлов и т.д. Рекурсивный алгоритм является самым медленным и его следует заменить на базовый или модифицированный.