

Министерство образования Российской Федерации
Московский Государственный Технический
Университет им. Н.Э. Баумана

Отчет по лабораторной работе №1
По курсу «Анализ алгоритмов»

Тема: «Алгоритм Левенштейна»

Студент: **Горохова И.Б.**
Группа: **ИУ7-51**

Преподаватель: **Волкова Л.Л.**

Москва, 2017

Содержание

Постановка задачи	3
Описание алгоритма	3
Реализация алгоритма	5
Примеры работы алгоритма	8
Заключение	9

Постановка задачи

В ходе лабораторной работы предстоит:

1. Изучить алгоритм Левенштейна
2. Реализовать алгоритм Левенштейна с использованием рекурсии, с использованием матрицы и модифицированный алгоритм на одном из языков программирования
3. Сравнить базовый и модифицированный алгоритмы Левенштейна

Описание алгоритма

Алгоритм Левенштейна - поиск минимального редакционного расстояния между двумя строками.

Результатом работы базового алгоритма Левенштейна является *минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.*

Допустимые редакторские 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.

Пример работы: Таблица преобразований слова **“WORD”** в слово **“QSORT”**.

R	I	M	M	R
W		O	R	D
Q	S	O	R	T

Минимальное редакционное расстояние - 3.

Рассчитать редакционное расстояние (Левенштейна) можно по рекуррентной формуле:

$$d(S_1, S_2) = D(M, N), \quad (1)$$

где M - длина строки S_1 , N - длина строки S_2

$$D(i, j) = \begin{cases} 0 & \text{if } i = 0, j = 0 \\ i & \text{if } i > 0, j = 0 \\ j & \text{if } i = 0, j > 0 \\ \min(D(i, j-1) + 1, \\ D(i-1, j) + 1, \\ D(i-1, j-1) + m(S_1[i], S_2[j])) & \text{if } i > 0, j > 0 \end{cases} \quad (2)$$

где $m(a, b) = 0$, если $a = b$, 1 - иначе

Использование матрицы для расчета редакционного расстояния:
Если длины строк S_1 и S_2 соответственно равны M и N , то найти расстояние Левенштейна можно, используя матрицу размерностью $(M + 1) * (N + 1)$:

	-	W	O	R	D
-	0	1	2	3	4
Q	1	1	2	3	4
S	2	2	2	3	4
O	3	3	2	3	4
R	4	4	3	2	3
T	5	5	4	3	3

Таблицу заполняют с ячейки (0,0).

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

Например, в ячейке (2,1) содержится значение 2, что значит, что преобразовать подстроку "QS" в подстроку "W" можно с помощью двух операций.

Модификация алгоритма

Модификация алгоритма Левенштейна состоит в добавлении операции транспозиции (перестановки) двух соседних символов к операциям вставки, удаления и замены. Например, редакционное расстояние между словами "WROD" и "WORD" с учетом модификации будет равно 1.

Реализация алгоритма

Листинг 1: Алгоритм Левенштейна с использованием рекуррентной формулы

```
1 int Recur(char *str1 , int M, char *str2 , int N)
2 {
3     return D(M,N, str1 , str2);
4 }
```

Входные данные: *str1* - первая строка, *M* - длина первой строки *str2* - вторая строка, *N* - длина второй строки.

Выходные данные: значение редакционного расстояния.

Листинг 2: Функция D

```
1 int D(int i , int j , char *str1 , char *str2)
2 {
3     if (i == 0 && j == 0)
4         return 0;
5     if (i > 0 && j == 0)
6         return i;
7     if (i == 0 && j > 0)
8         return j;
9     if (j > 0 && i > 0)
10    {
11        //delete [i]
12        int a = D(i , j - 1, str1 , str2) + 1;
13        //insert [j]
14        int b = D(i - 1, j , str1 , str2) + 1;
15        //replace [i] to [j]
16        int c = D(i - 1, j - 1, str1 , str2) + m(str1[i - 1], str2[j - 1]);
17        return min(a, b, c);
18    }
19 }
```

Входные данные: *i* - номер символа в первой строке, *j* - номер символа во второй строке, *str1*, *str2* - строки.

Листинг 3: Функция m

```
1 int m(char c1 , char c2)
2 {
3     return (c1 == c2) ? 0 : 1;
4 }
```

Листинг 4: Функция min

```
1 int min (int a, int b, int c)
2 {
3     return (a < b) ? ((a < c) ? a : c) : ((b < c) ? b : c);
4 }
```

Листинг 5: Алгоритм Левенштейна с использованием матрицы

```
1 int BaseMatrix(char *str1, int M, char *str2, int N)
2 {
3     //allocate memory
4     int** matrix = allocate_memory(M+1, N+1);
5     //filling zero row and column
6     for (int i = 0; i < M+1; i++)
7         matrix[i][0] = i;
8     for (int i = 0; i < N+1; i++)
9         matrix[0][i] = i;
10    //filling matrix
11    for (int i = 1; i < M+1; i++)
12        for (int j = 1; j < N+1; j++)
13            matrix[i][j] = min(matrix[i-1][j]+1, matrix[i][j-1] +
14                               1,
15                               matrix[i-1][j-1] + m(str1[i-1], str2[j-1]))
16
17    int D = matrix[M][N];
18    //free memory
19    free(matrix);
20    return D;
21 }
```

Листинг 6: Модифицированный алгоритм Левенштейна с использованием матрицы

```
1 int ModifMatrix(char *str1, int M, char *str2, int N)
2 {
3     unsigned long long int time1;
4     int** matrix = allocate_memory(M+1, N+1);
5     time1 = tick();
6     for (int i = 0; i < M+1; i++)
7         matrix[i][0] = i;
8     for (int i = 0; i < N+1; i++)
9         matrix[0][i] = i;
10 }
```

```

11  for (int i = 1; i < M+1; i++)
12      for (int j = 1; j < N+1; j++)
13      {
14          if (i > 1 && j > 1 && str1[i-1] == str2[j-2] && str1[
15              i-2] == str2[j-1])
16              matrix[i][j] = min4(matrix[i-1][j]+1, matrix[i][j
17                  -1] + 1,
18                  matrix[i-1][j-1] + m(str1[i-1], str2[j-1]),
19                  matrix[i-2][j-2]+1);
20          else
21              matrix[i][j] = min(matrix[i-1][j]+1, matrix[i][j-1]
22                  + 1,
23                  matrix[i-1][j-1] + m(str1[i-1], str2[j
24                      -1])));
25      }
26      printf("[TIME] ModifMatrix algorithm (%d*%d): %llu\n", M,
27          N, tick()-time1);
28      #if(0)
29      if (M < 8 && N < 8)
30          print_matrix(M+1, N+1, matrix);
31      #endif
32
33      int D = matrix[M][N];
34      free(matrix);
35      return D;
36  }

```

Входные данные: str1 - первая строка, M - длина первой строки str2 - вторая строка, N - длина второй строки.

Выходные данные: значение редакционного расстояния.

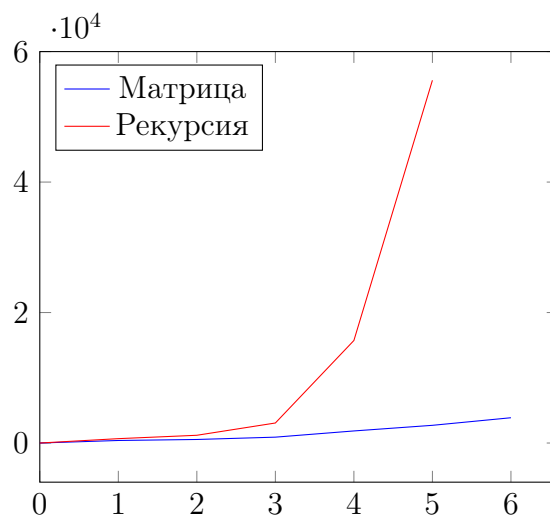
Функция *allocate_{memory}(m, n)* выделяет память под матрицу размерностью m*n. Функция *free(matrix)* освобождает память из под матрицы matrix.

Примеры работы алгоритма

Первая строка	Вторая строка	Базовый алгоритм	Модифицированный алгоритм	Тест
Word	word	1	1	Замена
word	world	1	1	Добавление
dog	urdoggy	4	4	Добавление в начале/конце
jiraffe	jiraffe	1	1	Удаление
word		4	4	Пустая строка
	word	4	4	Пустая строка
a b c	a b c	0	0	Одинаковые строки
luck	ulck	2	1	Перестановка
mother	omthred	5	3	Перестановка

Время работы алгоритмов

Время работы алгоритмов на графике представлено средним значением из пяти замеров:



Так как количество вызовов функции в рекурсивном алгоритме зависит от длины использованных строк, то чем больше длины строк, тем дольше работает алгоритм. Причем функция для одних и тех же параметров может вызываться несколько раз.

Время работы алгоритма, использующего матрицу, намного меньше рекурсивного алгоритма благодаря тому, что в нем требуется только $(m + 1) * (n + 1)$ операций заполнения ячейки матрицы.

Работа базового и модифицированного алгоритмов

Расчет редакционного расстояния между “**REESRE**” и “**REVERSE**”

	Базовый алгоритм		Модифицированный алгоритм	
Операции	R->R	M	R->R	M
	E->E	M	E->E	M
	->V	I	->V	I
	E->E	M	E->E	M
	S->R	R	S->R	C
	R->S	R	E->E	M
	E->E	M		
Результат	3 операции		2 операции	

Заключение

В ходе лабораторной работы я изучила алгоритм Левенштейна, реализовала алгоритм Левенштейна с использованием рекурсии, с использованием матрицы и модифицированный алгоритм на языке программирования C, сравнила базовый и модифицированный алгоритмы Левенштейна.