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

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:

$$d(S_1, S_2) = D(M, N), \tag{1}$$

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

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

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

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

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

```
int D(int i, int j, char *str1, char *str2)
{
  if (i == 0 && j == 0)
    return 0;
  if (i > 0 && j == 0)
    return i;
  if (i == 0 && j > 0)
    return j;
  if (j > 0 && i > 0)
  {
    //delete [i]
    int a = D(i,j-1,str1,str2)+1;
    //insert [j]
    int b = D(i-1,j,str1,str2)+1;
    //replace [i] to [j]
    int c = D(i-1, j-1,str1,str2)+m(str1[i-1], str2[j-1]);
    return min(a,b,c);
  }
}
```

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

Листинг 3: Функция m

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

Листинг 4: Функция min

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

Листинг 5: Алгоритм Левенштейна с использованием матрицы

```
int BaseMatrix(char *str1, int M, char *str2, int N)
{
  //allocate memory
  int** matrix = allocate_memory(M+1, N+1);
  //filling zero row and column
  for (int i = 0; i < M+1; i++)
    matrix[i][0] = i;
  for (int i = 0; i < N+1; i++)
    matrix[0][i] = i;
  //filling matrix
  for (int i = 1; i < M+1; i++)
    for (int j = 1; j < N+1; j++)
      matrix[i][j] = min(matrix[i-1][j]+1, matrix[i][j-1] +
          1,
                matrix[i-1][j-1] + m(str1[i-1], str2[j-1]))
                    ;

  int D = matrix[M][N];
  //free memory
  free(matrix);
  return D;
}
```

Листинг 6: Модифицированный алгоритм Левенштейна с использованием матрицы

```
int ModifMatrix(char *str1, int M, char *str2, int N)
{
  unsigned long long int time1;
  int** matrix = allocate_memory(M+1, N+1);
  time1 = tick();
  for (int i = 0; i < M+1; i++)
    matrix[i][0] = i;
  for (int i = 0; i < N+1; i++)
    matrix[0][i] = i;

```

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

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

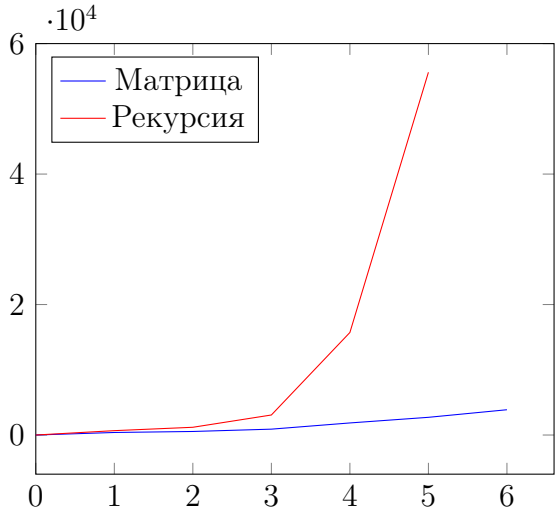
Функция $allocate_memory(m, n)$ выделяет память под матрицу размерностью m*n. Функция $free(matrix)$ освобождает память из под матрицы matrix.

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
jirafffe	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 значение 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.