МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Вычислительная техника»

Дисциплина «Геометрическое моделирование»

**Лабораторная работа №3.**

**Алгоритмы вычисления расстояния между строками**

**3 вариант**

Выполнил:

студент группы ИВТАПбд-31.

Вершинин Д. В.

Проверил:

Войт Н. Н.

Ульяновск, 2018

**Техническое задание:** Вычислить редакционное расстояние между двумя строками используя алгоритм Хиршберга.

**Ход работы:** Редакционное расстояние между двумя строками в теории информации и компьютерной лингвистике — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Расстояние Левенштейна и его обобщения активно применяется:

- для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи).

- для сравнения текстовых файлов утилитой diff и ей подобными. Здесь роль «символов» играют строки, а роль «строк» — файлы.

- в биоинформатике для сравнения генов, хромосом и белков.

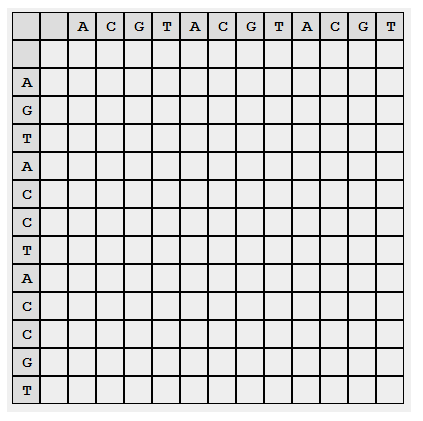
Формальная постановка задачи звучит следующим образом. Пусть имеется две строки S1 и S2. Мы хотим перевести одну в другую (пусть первую во вторую, легко показать, что операции симметричны), используя следующие операции:

- I: Вставка символа в произвольное место;

- D: Удаление символа с произвольной позиции;

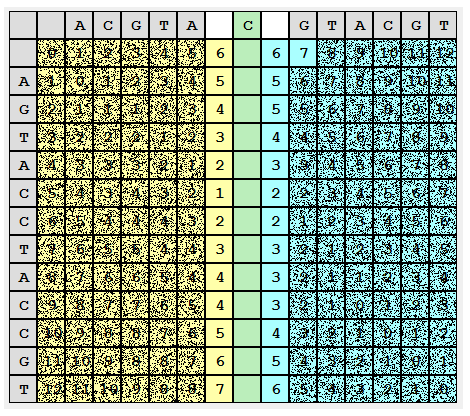
- R: Замена символа на другой.

Тогда d(S1,S2) — минимальное количество операций I/D/R для перевода S1 в S2. Эту величину и требуется найти. Для нахождения воспользуемся рекурсивным алгоритмом Хиршберга. Пусть требуется найти редакционное расстояние между строками S1 = ACGTACGTACGT и  
S2 = AGTACCTACCGT. На первом шаге алгоритма строится матрица размером |S1|\*|S2|. В каждой ячейке таблицы будет находиться значение редакционного расстояния, d[i][j] — минимальная стоимость получения префикса S2[0..(j - 1)] из префикса S1[0..(i - 1)].

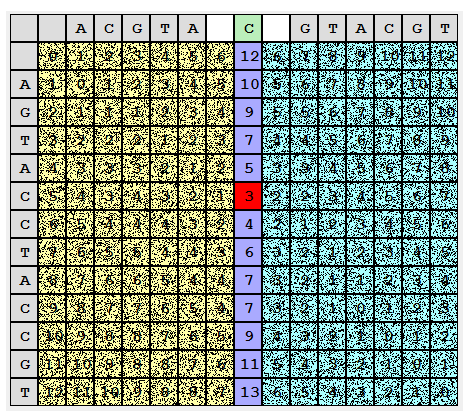


Матрица будет заполняться также как и в алгоритме Вагнера-Фишера по следующей рекуррентной формуле – d[i][j] = min(d[i][j - 1] + 1, d[i - 1][j] + 1, d[i - 1][j - 1] + X), где X = 1, если A[i - 1] ≠ B[j - 1], иначе X = 0. База рекурсии: d[0][0] = 0, d[i][0] = i, d[0][j] = j. То есть на каждом шаге заполнения таблицы мы выбираем, что выгоднее: удалить символ (D(i-1, j)), добавить (D(i, j-1)), или заменить (D(i-1, j-1)).

Поделим исходную задачу на подзадачу, поделив вторую строку на две равные (или почти равные) части, запомним место разбиения, и начнем заполнять матрицу по правилам, указанным ранее сверху вниз, слева направо. Вторую часть разбиения будем заполнять аналогично, но уже снизу-вверх, справа налево. Для удобства и сокращение размера кода будем заполнять матрицу точно так же, но для перевернутых строк, после чего перевернем последний столбец.



Теперь можно получить часть решения общей задачи, совместив частные. Просуммируем получившиеся в левых и правых столбцах значения и выберем минимум.



Что означает это совмещение? Мы попытались совместить первую строку с первой половиной второй строки, затем первую строку с второй половиной строки. Сумма позволяет получить количество операций изменения первой строки, чтобы привести ее к конкатенации первой и второй половины второй строки. теперь мы знаем как «разрезать» первую строку на две половины, чтобы соответствующие пары дали нам редакционное расстояние: d(S11, S21) + d(S12, S22) = d(S1, S2). Строка (в матрице), в которой расположен минимум и есть искомое разбиение S1. Аналогично будем делить задачу уже для получившихся подстрок S11, S21 и S12, S22. Если длина подстроки = 0, задача тривиальна.

Анализ алгоритма: для каждого шага потребуется запоминать не более чем один столбец d(i,\*) и набор разбиений S1. Итого O(|S1| + |S2|). каждое разбиение S[0..N] по столбцу t порождает обработку подстрок S[0..t] и S[t..N]. Всего разбиений N. При обработке каждого разбиения (S1[i..j], S1[k..n]) требуется (j — i) x (n — k) операций. Суммируя оценку худших случаев для подстрок получаем в итоге 2 \* |S1| \* |S2| операций.

**Исходный код алгоритма:**

#include <stdio.h>

#include <iostream>

#include <algorithm>

#include <string>

#include <vector>

using namespace std;

int d[1000][1000];

vector<int> calcD(string a, string b) {

for (int i = 0; i <= 1000; i++) { //очищаем матрицу

for (int j = 0; j <= 100; j++) {

d[i][j] = 0;

}

}

for (int i = 0; i <= a.size(); i++) { //база рекурсии

d[i][0] = i;

}

for (int i = 0; i <= b.size(); i++) {

d[0][i] = i;

}

for (int i = 1; i <= a.length(); i++) { //общий случай рекурсии

for (int j = 1; j <= b.length(); j++) {

int x;

a[i - 1] != b[j - 1] ? x = 1 : x = 0;

d[i][j] = min(d[i][j - 1] + 1, min(d[i - 1][j] + 1, d[i - 1][j - 1] + x));

}

}

vector<int> lastLine; //возвращаем последний столбец заполненной матрицы

for (int j = 0; j <= a.length(); j++)

lastLine.push\_back(d[j][b.length()]);

return lastLine;

}

int Hirschberg(string s1, string s2) {

string s1l, s2l, s1r, s2r;

if (s1.length() == 0 || s2.length() == 0) //проверяем тривиальные случаи

return max(s1.length(), s2.length());

if (s1.length() == 1 || s2.length() == 1) {

if (s1.length() < s2.length())

if (s2.find(s1[0]) != string::npos)

return s2.length() - 1;

else

return s2.length();

else

if (s1.find(s2[0]) != string::npos)

return s1.length() - 1;

else

return s1.length();

}

if (s2.length() < s1.length()) //для удобства будем всегда работать со второй строкой

swap(s1, s2);

int left, right;

if (s2.length() % 2 == 0) { //разбиваем строку на 2 равные подстроки

left = s2.length() / 2;

right = s2.length() / 2;

}

else {

left = (s2.length() - 1) / 2;

right = (s2.length() + 1) / 2;

}

s2l = s2.substr(0, left);

s2r = s2.substr(s2.length() / 2, right);

vector<int> e = calcD(s1, s2l); //вычисляем последний столбец в матрице Вагнера-Фишера (считаем d(s1, s2l))

reverse(s2r.begin(), s2r.end());

reverse(s1.begin(), s1.end());

vector<int> d = calcD(s1, s2r); //тоже делаем для правой половины второй строки

reverse(s2r.begin(), s2r.end());

reverse(s1.begin(), s1.end());

reverse(d.begin(), d.end());

vector<int> sum;

int minind = 0;

for (int i = 0; i < e.size(); i++) { //вычисляем индекс минимальной суммы редакционных расстояний

sum.push\_back(e[i] + d[i]);

if (sum[i] < sum[minind])

minind = i; //минимальный индекс будет являться индексом разбиения

}

s1l = s1.substr(0, minind); // разбиваем первую строку на две подстроки

s1r = s1.substr(minind, s1.length() - minind);

return Hirschberg(s1l, s2l) + Hirschberg(s1r, s2r);

}

int main() {

string a = "AGTACCTACCGT", b = "ACGTACGTACGT";

cout << Hirschberg(a, b);

}