Алгоритм Вагнера-Фишера: визуализация и краткое описание.

Алгоритм позволяет вычислить расстояние Левенштейна между строками. Похожесть исходной и целевой строки измеряется как количество замен, вставки и удаления, необходимых для преобразования одной строки в другую. Расстояние Левенштейна (LD) увеличивается в зависимости от количества преобразований, необходимых для превращения одной строки в другую.

Если исходная строка – «тип», а целевая – «топ», то LD = 1. Это означает, что нам необходимо сделать одну замену, для того чтобы преобразовать одну строку в другую.

Рассмотрим работу алгоритма на простом примере. Рассчитаем расстояние Левенштейна между “КОРОВА” и “СОЛОМА”. Составим таблицу.

		К	О	Р	О	В	А
	0	1	2	3	4	5	6
С	1						
О	2						
Л	3						
О	4						
М	5						
А	6						

Начнём с колонки “К” сверху вниз, затем “О” и т.д.. Нам необходимо сравнивать символ в колонке с символом в строке. Если они совпадают, то мы просто кладём в

текущую ячейку значение из (i-1,j-1). Если они не совпадают, тогда нужно выбрать минимум из трёх значений, увеличенных на единицу.

		К	О	Р	О	В	А
	0	1	2	3	4	5	6
С	1	1	2	3	4	5	6
О	2	2	1	2	3	4	5
Л	3	3	2	2	3	4	5
О	4	4	3	3	2	3	4
М	5	5	4	4	3	3	4
А	6	6	5	5	4	4	**3**

Заполняя таблицу по этому алгоритму, получаем в последней ячейке значение расстояния LD равное 3.

Этот алгоритм можно применить для анализа схожести исходных кодов программ. Проведя все этапы подготовки, получаем последовательность токенов для анализа. Процент схожести можно рассчитать по формуле: P = (1-Diff/(maxLen(S1,S2))*100.

Продемонстрируем работу алгоритма на реальном коде:

Program 1	Program 2

```cpp
#include <iostream>
using namespace std;
void bubble(int* arr, int size){
    for (int i = size = 1; i>=0; i--){
            for (int j = 0; j < i ;j++){
                if (arr[j]>arr[j+1]){
                    int tmp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1]=tmp;
                }
            }
    }
}

int main() {
    int arr[]  = {3, 2, 1, 5, 4 , 22, 31, 0, -3, -5, 1, 3};
    int size = 12;
    bubble(arr, size);
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
void bubble(double* arr, int size){
    int i =size;
    while (i>=0){
        for (int j = 0; j < i ;j++){
            if (arr[j]>arr[j+1]){
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1]=tmp;
            }
        }
        i--;
    }
}

int main() {
    int size,i;
    double arr[]  = {3, 2, 1, 5, 4 , 22, 31, 0, -3, -5, 1, 3};
    size = 12;
    i=0;
    bubble(arr, size);
    for (i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Модифицированный код имеет следующие отличия:

- Изменены типы входных аргументов
- Один из циклов заменён на while
- Добавлен некоторый “шум”

Внутреннее представление кода:

1. IOIOIOIOKKIKIVOIVICVIOIONIONIOCVIONIOIIOKIIOOIIONO VIOIIOIIOOIIONOIIONOOIVIVIOONNNNNNNNONONNNVIO NIIICVIONIOIOIIOIIOOKN
2. IOIOIOIOKKIKIVOIVIVIOICIONCVIONIOIIOKIIOOIIONO VIOIIOIIOOIIONOIIONOOI**IO**VI**VII**VIOONNNNNNNN**ION**ONO NNNVIONIIICVIONIOIOIIOIIOOKN

I - IDENTIFICATOR
O - OPERATOR
V - VARIABLETYPE
C - CYRCLE
N - NUMBER

Из примера получаем LD равное 13 и процент совпадение равное 74%.
Таким образом можно подвести некое резюме по алгоритму в контексте антиплагиата:

1. Уязвимость перед перестановкой местами блоков кода
2. Относительная устойчивость перед добавлением “шума” в программу(получим незначительное понижение процента).
4. Асимптотика **O**(M*N) по времени и по памяти.