

Государственное образовательное учреждение высшего профессионального образования

Москва 2018

Содержание

Введение	3
1 Аналитический раздел	4
1.1 Описание Алгоритмов	4
1.1.1 Расстояние Левенштейна	4
1.1.2 Расстояние Дамерау – Левенштейна	5
2 Конструкторский раздел	7
2.1 Разработка алгоритмов	7
2.1.1 Расстояние Левенштейна(обычный)	7

Введение

Целью работы является изучение и применение метода динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна, а так же реализовать алгоритм Левенштейна в рекурсивном виде. Для достижения поставленной цели необходимо решить следующие задачи:

- Изучить алгоритмы Левенштейна и Дамера-Левенштейна нахождения расстояния между строками;
- Применить метод динамического программирования для матричной реализации указанных алгоритмов;
- Получить практические навыки реализации указанных алгоритмов: двух алгоритмов в матричной версии и алгоритма Левенштейна, реализованного рекурсивно;
- Провести сравнительный анализ линейной и рекурсивной реализаций алгоритма Левенштейна по затрачиваемым ресурсам(времени и памяти);
- Привести экспериментальное подтверждение различий во временной эффективности рекурсивной и нерекурсивной реализаций алгоритма Левенштейна при помощи разработанного ПО на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк;
- Описать и обосновать полученные результаты о выполненной работе;

1 Аналитический раздел

Перед теоритическим изложением алгоритмов, представленных в работе, требуется ввести понятия *редукционного расстояния* и *метода динамического программирования*.

Редукционное расстояние(*расстояние Эйнштейна*) – это минимальное количество редукционных операций, необходимых для преобразования одной строки в другую.

Есть следующие редукционные операции:

- Операции, вес которых - 1:
 - I - insert(вставка);
 - D - delete(удаление);
 - R - replace(замена);
- Операция, вес которой - 0:
 - M - match(совпадение);

Так минимальное расстояние между строками $\min D(\text{'увлечение'}, \text{'развлечение'}) = 3$, но чтобы найти это, требуется перебрать расстояния с разным выравниваем строк по отношению друг к другу.

		у	в	л	е	ч	е	н	и	е
р	а	з	в	л	е	ч	е	н	и	е
I	I	R	M	M	M	M	M	M	M	M

Проблема выравнивания решается рекуррентно через расстояния между подстроками фиксированной длины.

1.1 Описание Алгоритмов

Первым появившимся алгоритмом был алгоритм Левенштейна, который заложил фундамент в поиске расстояния между строками.

1.1.1 Расстояние Левенштейна

Расстояние Левенштейна имеет широкую область применения. Алгоритм используется в:

- поисковых системах, в базах данных, в автоматическом распознавание текста и речи для исправления ошибок и опечаток в слове;
- в утилитах для сравнения файлов(таких как diff);
- в биоинформатике для сравнения генов, хромосом и белков;

Алгоритм рекуррентно через расстояния между подстроками i и j находит расстояние между строками $s1$ и $s2$. Проверяя, какое действие будет наиболее выгодным $I(\text{insert})$, $D(\text{delete})$, $M(\text{match})$ или $R(\text{replace})$:

$$D(s1, s2) = D(s1[1..i], s2[1..j]) = \min \left(\begin{array}{l} D(s1[1..i], s2[1..j-1]) + 1, \\ D(s1[1..i-1], s2[1..j]) + 1, \\ D(s1[1..i-1], s2[1..j-1]) + \left[\begin{array}{l} 0, \text{ если } s1[i] = s2[j], \\ 1, \text{ иначе} \end{array} \right] \end{array} \right) \quad (1.1)$$

, где i – длина подстроки строки $s1$, которая изначально равно длине $s1$, j – длина подстроки строки $s2$, которая изначально равно длине $s2$

Таким образом, применив метод динамического программирования мы разбиваем нашу задачу на небольшие подзадачи, которые достаточно легко можно решить. Данный метод удобно представлять в виде матрицы.

	λ	M	Γ
λ	0	1	2
M	0	0	1

¹

В виде матрицы редукционные операции можно представить следующим образом:

- $I(\text{insert}) \rightarrow$;
- $D(\text{delete}) \downarrow$;
- $M(\text{match})$ or $R(\text{replace}) \searrow$;

У данного алгоритма есть улучшенная версия, которую модифицировал Фредерик Дамерау.

1.1.2 Расстояние Дамерау – Левенштейна

Данный алгоритм применяется также как и обычный в:

- поисковых системах;
- биоинформатике(сравнение белков линейной структуры);

Причиной появления данного алгоритма было огромное количество ошибок ввода – ввод двух соседних символов не в том порядке. Отсюда появляется новая операция в дополнении к уже имеющимся:

- X - exchange(или T - transposition);

¹(Представление $D('M', 'MG')$)

Отсюда, формула 1.1 переходит в:

$$D(s1[1..i], s2[1..j]) = \min \left(\begin{array}{c} D(s1[1..i], s2[1..j-1]) + 1, \\ D(s1[1..i-1], s2[1..j]) + 1, \\ D(s1[1..i-1], s2[1..j-1]) + \begin{cases} 0, \text{ если } s1[i] = s2[j], \\ 1, \text{ иначе} \end{cases}, \\ D(s1[1..i-2], s2[1..j-2]) + 1 \end{array} \right), \quad (1.2)$$

Причем последняя операция(X) выполняется, если такая перестановка необходима и требуется, и это не совпадение.

2 Конструкторский раздел

Ниже представлены схемы алгоритмов – Дамерау - Левенштейна и двух реализаций Левенштейна(рекурсивный и обычный).

2.1 Разработка алгоритмов

2.1.1 Расстояние Левенштейна(обычный)

subprocedures

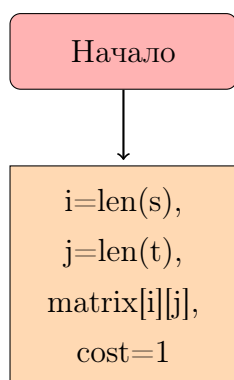
DoAE: direction of arrival estimation

UID : user identification

DoAT: DoA tracking

BF : beam forming

SF : spatial filtering



variables

DoA : direction of arrival

i : identification sequence

X, x : signal model

DoA_T: DoAs up to date

$\hat{x}(t)$: filtered signal