УДК 004.85
DOI: 10.53015/18159958_2024_20_1_61

ПРИМЕНЕНИЕ АЛГОРИТМОВ НЕЧЕТКОГО СРАВНЕНИЯ ДЛЯ РАСПОЗНАВАНИЯ КОМАНД

APPLICATION OF THE FUZZY COMPARISON ALGORITHMS FOR COMMAND RECOGNITION

©2024

Седых Ирина Александровна[1], доктор технических наук, профессор кафедры автоматизированных систем управления
Скорина Тарас Александрович, студент
Sedykh Irina Alexandrovna[1], Doctor of Technical Sciences, professor of the Department of Automated control systems
Skorina Taras Alexandrovich, student

Липецкий государственный технический университет, Липецк (Россия)
Lipetsk State Technical University, Lipetsk (Russia)

E-mail: sedykh-irina@yandex.ru [1]ORCID: https://orcid.org/0000-0003-0012-8103
E-mail: duce.skoriny@yandex.ru

Аннотация: В данной работе рассматривается применение алгоритмов нечеткого сравнения строк для задачи распознавания команд. Описываются три алгоритма: Левенштейна, Дамерау-Левенштейна и Джаро-Винклера. Приводится их математическое описание и особенности применения. Проводится оценка точности работы алгоритмов на наборе тестовых данных.

Ключевые слова: нечеткое сравнение строк, распознавание команд, редакционное расстояние, алгоритм Левенштейна, алгоритм Дамерау-Левенштейна, алгоритм Джаро-Винклера.

Для цитирования: Седых И.А., Скорина Т.А. Применение алгоритма нечеткого сравнения для распознавания команд // Вести высших учебных заведений Черноземья. – 2024. – Т. 20, № 1 (75). – С. 61–70. DOI: 10.53015/18159958_2024_20_1_61.

Abstract: The article discusses the application of fuzzy string comparison algorithms for the task of command recognition is considered. Three algorithms are described: Levenstein, Damerau-Levenstein and Jaro-Winkler. Their mathematical description and application features are given. The accuracy of the algorithms is evaluated on a set of test data.

Keywords: fuzzy string comparison, command recognition, editorial distance, Levenshtein algorithm, Damerau-Levenshtein algorithm, Jaro-Winkler algorithm.

For citation: Sedykh I.A., Skorina T.A. Application of the fuzzy comparison algorithm for command recognition // Vesti of Higher Educational Institutions of the Chernozem region. – 2024. – Vol. 20, № 1 (75). – P. 61–70. DOI: 10.53015/18159958_2024_20_1_61.

ВВЕДЕНИЕ

Распознавание команд является важной задачей во многих интерактивных системах, таких как голосовые помощники, интерфейсы командной строки и т.д. Зачастую пользовательский ввод может содержать ошибки или отклонения от ожидаемой формы команды, затрудняющие ее распознавание. Одним из подходов к решению этой проблемы является применение алгоритмов нечеткого сравнения строк, которые позволяют гибко сопоставлять команды с учетом возможных ошибок и вариаций ввода.

В данной работе рассматриваются три алгоритма нечеткого сравнения строк: алгоритм вычисления расстояния Левенштейна, алгоритм вычисления расстояния Дамерау-Левенштейна и алгоритм схожести Джаро-Винклера. Приводится их математическое описание, особенности применения и сравнительный анализ эффективности для задачи распознавания команд на наборе тестовых данных с искажениями.

ПРИМЕНЕНИЕ АЛГОРИТМА НЕЧЕТКОГО СРАВНЕНИЯ ДЛЯ РАСПОЗНАВАНИЯ КОМАНД

Нечёткое сравнение строк может определено и основано на вычислениях. Эта концепция обычно называется строковой метрикой и представляет собой метод выражения в числах того, насколько похожи две строки. Существует множество показателей различной степени сложности и полезности. Традиционным примером строковых метрик является расстояние Левенштейна.

РАССТОЯНИЕ ЛЕВЕНШТЕЙНА

Расстояние Левенштейна, также известное как редакционное расстояние, представляет собой метрику для измерения различий между двумя последовательностями символов путем оценки числа необходимых операций для преобразования одной строки в другую. Классический алгоритм нахождения расстояния Левенштейна предполагает оценку трёх операций: вставки, удаления и замены символа [1]. Чем меньше редакционное расстояние, тем более похожи строки. Для двух строк s и t длиной |s| и |t| соответственно расстояние $d(s,t)$ определяется следующим образом:

$$d(s,t) = \min\Big(d[i-1,j]+1, d[i,j-1]+1, d[i-1,j-1]+\delta_{s_i \neq t_j}\Big), \quad (1)$$

где $d[i,j]$ – расстояние между подстроками $s[1..i]$ и $t[1..i]$; $\delta_{s_i \neq t_j}$ – функция, возвращающая 0, если символы s_i и t_j совпадают, и 1 в противном случае:

$$\delta_{s_i \neq t_j} = \begin{cases} 0, \text{если } s_i = t_j \\ 1, \text{если } s_i \neq t_j \end{cases}. \quad (2)$$

АЛГОРИТМ ВАГНЕРА-ФИШЕРА

Для вычисления наименьшего расстояния осуществляется расчёт матрицы D с использованием формулы (1). Матрица имеет следующий вид, представленный в таблице 1.

Существуют различные реализации данного алгоритма на языке Python [2]. Представленный далее вариант реализации учитывает возможность задания произвольных цен для вставок, удалений и замен:

```
def wagner_fischer(S1, S2, cost_insert, cost_delete, cost_replace):
    M = len(S1)
    N = len(S2)
    D = [[0] * (N + 1) for _ in range(M + 1)]

    # Инициализация первого столбца и первой строки
    for i in range(1, M + 1):
        D[i][0] = D[i - 1][0] + cost_delete(S1[i - 1])
      for j in range(1, N + 1):
        D[0][j] = D[0][j - 1] + cost_insert(S2[j - 1])
```

```
    for i in range(1, M + 1):
        for j in range(1, N + 1):
            cost_del = D[i - 1][j] + cost_delete(S1[i - 1])
            cost_ins = D[i][j - 1] + cost_insert(S2[j - 1])
            cost_rep = D[i - 1][j - 1] + cost_replace(S1[i - 1], S2[j -
1])
            D[i][j] = min(cost_del, cost_ins, cost_rep)

    return D[M][N]
```

Таблица 1

Таблица Вагнера-Фишера для сравнения строк

Table 1

Wagner-Fischer table for string comparison

		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	7
L	11	10	9	8	9	8	8	8	8	7	6

РАССТОЯНИЕ ДАМЕРАУ-ЛЕВЕНШТЕЙНА

Расстояние Дамерау-Левенштейна представляет собой модификацию алгоритма Левенштейна, допускающую операции транспозиции (перестановки соседних символов). Это может быть полезно при распознавании команд, где часто встречаются опечатки, связанные с неправильным порядком символов. В данной реализации есть как и плюсы, так и минусы. Алгоритм позволяет сравнивать строки различной длины, однако есть недостаток в виде долгого времени исполнения, которое при сравнении слова со словарем существенно усложняет анализ больших текстов. Формула для расчета расстояния Дамерау-Левенштейна выглядит следующим образом:

$$d(s,t) = \min\Big(d[i-1,j]+1, d[i,j-1]+1, d[i-1,j-1]+\delta_{s_i \neq t_j}, d[i-2,j-2]+1+\delta_t(s_i,t_j,s_{i-1},t_{j-1})\Big), \quad (3)$$

где $\delta_t(s_i,t_j,s_{i-1},t_{j-1})$ – операция транспозиции, добавляющая 1 к расстоянию только в случае, если пары символов s_i и t_{j-1} и s_{i-1} и t_j равны.

$$\delta_t(s_i,t_j,s_{i-1},t_{j-1}) = \begin{cases} 1, \text{если } s_i = t_{j-1} \text{ и } s_{i-1} = t_j \\ 0, \text{иначе} \end{cases}, \quad (4)$$

При этом следует учитывать, что транспозиция обычно считается нежелательной и более "дорогостоящей" операцией, чем остальные [3, 4].

РАЗЛИЧИЯ РАССТОЯНИЙ ЛЕВЕНШТЕЙНА И ДАМЕРАУ-ЛЕВЕНШТЕЙНА

Для строк A = "a cat" и B = "an act" расстояние Левенштейна будет равно 3. Приведение строки A к строке B предполагает вставку символа (n) и две замены (“a” на “c” и “c” на “a”). Учитывая вышесказанное, таблица Вагнера-Фишера для данного случая будет иметь вид, представленный в таблице 2.

Последняя ячейка содержит общую стоимость 3. С добавлением операции перестановки таблица примет вид, представленный в таблице 3.

Таблица 2

Таблица Вагнера-Фишера для строк A и B

Table 2

Wagner-Fischer table for A and B strings

		a	n		a	c	t
	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
	2	1	1	1	2	3	4
c	3	2	2	2	2	3	3
a	4	3	3	3	2	3	3
t	5	4	4	4	3	3	3

Таблица 3

Таблица Вагнера-Фишера для расстояния Дамерау-Левенштейна

Table 3

Wagner-Fischer table for the Damerau-Lowenstein distance

		a	n		a	c	t
	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
	2	1	1	1	2	3	4
c	3	2	2	2	2	3	3
a	4	3	3	3	2	2	3
t	5	4	4	4	3	3	2

Заострить внимание следует на нескольких фактах:

- Последняя ячейка теперь имеет значение 2;
- Ячейка на пересечении ("a", "c") также имеет значение 2, потому что алгоритм распознает перестановку;
- Ячейка на пересечении пробелов является основой для стоимости ячейки на пересечении ("a", "c").

В ячейке ("a", "c") стоимость перестановки рассчитывалась путем возвращения к ячейке на пересечении пробелов, что представляет собой общую стоимость до перестановки, и добавления единицы к этому значению. Технически алгоритм отступает до точки перед перестановкой и проверяет, дает ли это меньшую стоимость для операций.

В этом случае предполагается хранение в памяти трёх строк вместо двух, как в традиционном алгоритме Вагнера-Фишера. Однако с точки зрения оптимизации это не является большой проблемой. Большей проблемой является предположение, что между переставленными символами не были вставлены или удалены символы. Тогда редакционное расстояние для строк "a

cat" и "an abct" возрастает до 4, хотя на самом деле достаточно трёх операций (вставка, перестановка, вставка). Аналогичным образом, от "a cat" до "a tc" расстояние составляет 3 при достаточных двух операциях (удаление, перестановка).

Следствием этого является невыполнение неравенства треугольника: расстояние между двумя последовательностями может быть больше суммы расстояний между каждой из них и третьей общей последовательностью.

Решение проблемы реализовано в алгоритме Дамерау-Левенштейна, который при вычислении перестановки осуществляет поиск последней строки с символом текущего абзаца и последнего столбца в той строке, где эти символы совпадали.

Если такие строка и столбец существуют, вместе они определяют ячейку C1. Далее фиксируется ячейка C2, расположенная в верхнем левом углу от текущей. Она выражает раннее состояние перед перестановкой. В случае существования операции перестановки, система возвращается к этому более раннему состоянию, а затем вычисляет стоимость возврата к текущему состоянию [5, 6].

Изменения значений не происходит, поскольку каждая ячейка может содержать правильное расстояние редактирования для подпоследовательностей. Вместо этого в рамках предположения о вставке и удалении подсчитывается количество символов между переставленными символами в A или B. Операции удаления считаются для A, а операции вставки для B. Меньший номер строки или столбца вычитается из большего минус один, чтобы исключить текущую ячейку из подсчета. Благодаря тому, что значение ячейки будет наименьшим из стоимостей вставки, удаления, замены или перестановки, выяснять, является ли операция конкретно вставкой или удалением, не требуется. Если ни одно из значений не является 0, стоимость будет слишком высокой для выбора. В рамках Python код будет иметь следующий вид:

```
matrix[last_match-
ing_row][last_match_col]
+ (row - last_matching_row - 1) +
1
+ (col - last_match_col - 1))
```

Когда перебор доходит до выделенной ячейки, алгоритм видит, что последний раз “a” встречалось в первой строке и в первом столбце (см. таблица 4). Поскольку это так, вычисляется стоимость текущей ячейки следующим образом:

- стоимость до перестановки (ячейка 1, 1): 0;
- расстояние до текущей строки: 3 (ячейка 4);
- расстояние до текущего столбца: 4 (ячейка 5);
- и 1 для самой перестановки.

Таким образом, редакционное расстояние Дамерау-Левенштейна 0 + 3 + 4 + 1 = 8.

Таблица 4

Таблица расстояния Дамерау-Левенштейна для «a cat» до «a abct»

Table 4

Damerau-Lowenstein location table for «a cat» to «a abct»

		a		a	b	c	t
	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
	2	1	0	1	2	3	4
c	3	2	1	1	2	2	3
a	4	3	2	1	2	2	3
t	5	4	3	2	2	3	2

АЛГОРИТМ ДЖАРО-ВИНКЛЕРА

Алгоритм Джаро-Винклера, в отличие от алгоритмов Левенштейна и Дамерау-Левенштейна, больше ориентирован на выявление похожих, но не идентичных строк. Он учитывает количество совпадающих символов и их порядок, но при этом допускает некоторые перестановки символов внутри строки [7, 8]. Зачастую этот алгоритм используется в задачах сопоставления идентификаторов. Сначала вычисляется расстояние Джаро:

$$\mathrm{Jaro}(s,t) = \frac{1}{3}\left(\frac{m}{|s|} + \frac{m}{|t|} + \frac{m-t}{m}\right), \qquad (5)$$

где m – количество общих символов между строками

$$\mathrm{JaroWinkler}(s,t) = \mathrm{Jaro}(s,t) + l \cdot p \cdot (1 - \mathrm{Jaro}(s,t)), \qquad (6)$$

где m – количество общих символов между строками, l – длина общего префикса строк s и t, p – коэффициент, который используется для увеличения значения расстояния в случае совпадения префикса.

При этом для измерения базовой степени сходства между строками сначала применяется функция $\mathrm{Jaro}(s,t)$, а после с помощью $\mathrm{JaroWinkler}(s,t)$ учитывается общий префикс строк и взаимное расположение символов.

На практике алгоритм демонстрирует проблему ложных транспозиций: при нахождении совпадающих символов в сравниваемых строках алгоритм некорректно считает наличие перестановок символов в тех случаях, когда в более длинной строке есть повторяющиеся символы, которые учитываются как транспозиции [9, 10].

Решением может являться ведение учёта уже найденных позиций совпадающих символов в более длинной строке. Перед записью позиции совпадающего символа производится проверка, не был ли этот символ на данной позиции уже найден ранее. Тогда поиск совпадений в более длинной строке продолжается, в противном случае текущая позиция фиксируется и происходит переход к следующему символу более короткой строки [11, 12].

Данный алгоритм может быть полезен при распознавании команд, когда ожидаются вариации в написании, сокращения или сленговые формы. Однако он менее эффективен при обработке команд, где точность написания имеет большое значение.

ОЦЕНКА ЭФФЕКТИВНОСТИ АЛГОРИТМОВ НЕЧЁТКОГО СРАВНЕНИЯ

Была проведена оценка точности работы алгоритмов. В качестве датасета использовался набор формата «ключ: значение» команд различной длины и их искажений, включающих перестановку слов, орфографические ошибки и частичную замену слов на синонимы [13]. Фрагмент проверочных данных представлен в таблице 5.

Таблица 5

Набор данных для проверки

Table 5

Test dataset

Ключ	Значение
воспроизведимзыку	воспроизвести музыку
кино режим	режим кино
наберинорем	набери номер
врубить свет	включить освещение
…	…

Таблица 6

Подбор наиболее близких вариантов распознавания для одной команды по трём алгоритмам

Table 6

Selection of the closest recognition options for one team using three algorithms

Команда\ Алгоритм	levenshtein	damerau_levenshtein	jarowinkler
'врубить свет'	включи свет (6)	включи свет (6)	включи свет (0.36)
	режим кино (9)	режим кино (9)	добавь в плейлист (0.37)
	отключи звук (10)	отключи звук (10)	увеличь громкость (0.38)

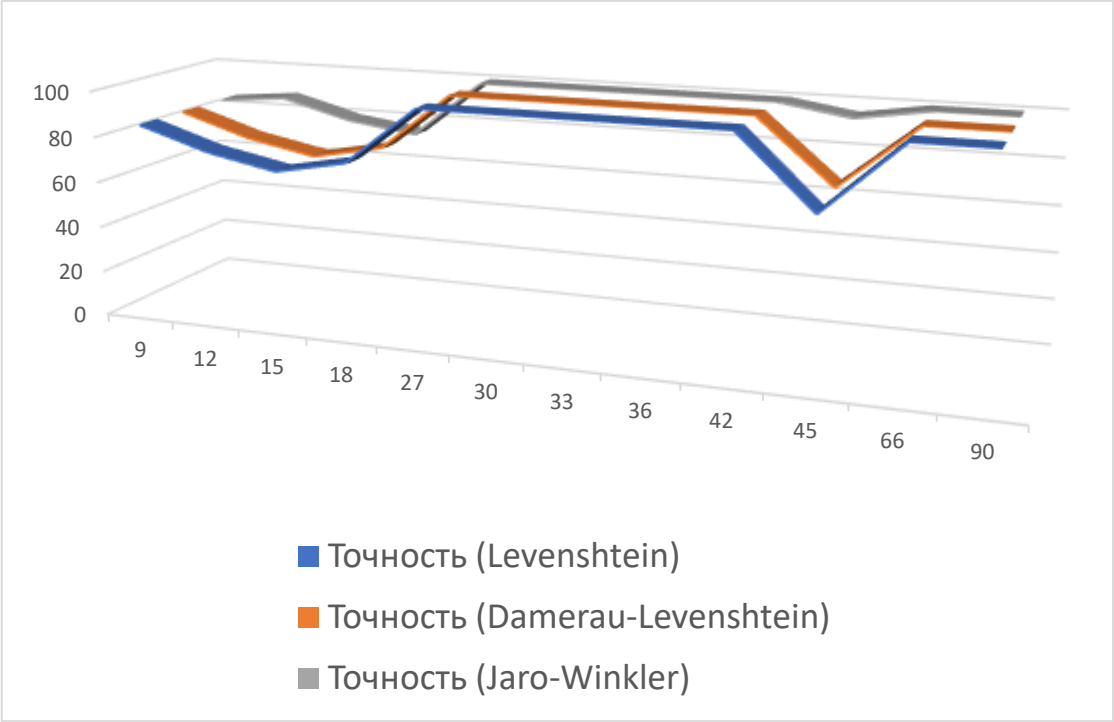


Рис. 1. Сравнение точности распознавания для трёх алгоритмов

Fig. 1. Comparison of recognition accuracy for three algorithms

Для каждой тестовой команды вычисляются расстояния и сходства с эталонными командами, подбираются наиболее близкие варианты (см. таблица 6). Стоит отметить, что для алгоритмов Левенштейна наиболее близким считается меньшее расстояние, а для Джаро-Винклера – большее сходство [14, 15].

Средняя точность для алгоритмов Левенштейна и Дамерау-Левенштейна составила около 76%, для Джаро-Винклера – 85%. На рисунке 1 представлен график динамики точности интерпретации с длиной проверяемых строк.

Следует отметить, что с основном команды составляют пары слов, иногда три, поэтому большая часть проверочных данных представлена ими. Наибольшую результативность показал алгоритм Джаро-Винклера, в том числе проявив устойчивость на длинных последовательностях.

ЗАКЛЮЧЕНИЕ

В работе были рассмотрены три алгоритма нечеткого сравнения строк: Левенштейна, Дамерау-Левенштейна и Джаро-Винклера - и их применение в задаче распознавания команд в интерактивных системах.

Проведена оценка эффективности работы всех трёх алгоритмов. Средняя точность для алгоритмов Левенштейна и Дамерау-Левенштейна составила около 76%, для алгоритма Джаро-Винклера - 85%.

Таким образом, алгоритм Джаро-Винклера продемонстрировал наибольшую эффективность для задачи распознавания команд с допущением вариаций их написания. Он показал высокую устойчивость, в том числе при работе с длинными последовательностями. Алгоритмы Левенштейна могут быть предпочтительнее в случаях, когда требуется строгое совпадение команд. Тем не менее, базовые строковые метрики не учитывают никакой семантической информации о строках. Они имеют дело исключительно с символами и просто измеряют, насколько похожи строки по количеству символов. Для эффективного применения необходима нормализация данные на небольшие атомарные атрибуты и выделение семантических компонентов. Строковые метрики не являются полным решением, но являются важной частью системы, которая может обеспечить нечеткое сопоставление, которая позволяет количественно оценить, насколько схожи компоненты объекта.

СПИСОК ЛИТЕРАТУРЫ

1. Левенштейн В.И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. АН СССР. 1965. 163 (4). С. 845–848.

2. Лещенко А.В. Практическое применение алгоритмов нечеткого поиска // Сборник научных трудов НГТУ. 2018. №3-4(93). С. 59-69.

3. Поиск нечетких соответствий: сравнение записей по расстоянию между строками [Электронный ресурс]. URL: https://www.megaputer.com/ru/fuzzy-matching-comparing-records-with-stringdistance-measures/ (дата обращения: 30.04.2024).

4. Wagner R. A., Fischer M. J. The string-to-string correction problem. J. ACM, v. 21(1), 1974. P. 168.

5. Damerau-Levenshtein Edit Distance Explained [Электронный ресурс]. URL:https://www.lemoda.net/text-fuzzy/damerau-levenshtein/ (дата обращения: 30.04.2024).

6. Jaro distance [Электронный ресурс]. URL: https://rosettacode.org/wiki/Jaro_distance(дата обращения: 30.04.2024).

7. Winkler W. E. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage, Proceedings of the Section on Survey Research Methods, American Statistical Association, 1990, P. 354–359.

8. Сходство Джаро - Винклера. Нечеткое сравнение строк [Электронный ресурс]. URL: http://xn---1-6kca8bgsjrjhe.xn--p1ai/public/1172479/ (дата обращения: 30.04.2024).

9. Code Spelunking: Jaro-Winkler String Comparison [Электронный ресурс]. URL: https://lingpipe-blog.com/2006/12/13/code-spelunking-jarowinkler-string-comparison/ (дата обращения: 30.04.2024).

10. Navarro G. A guided tour to approximate string matching, ACM Computing Surveys, 2001, v.33 (1), P. 31–88.

11. Winkler W. E. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage, Proceedings of the Section on Survey Research Methods, American Statistical Association, 1990, P. 354–359.

12. Brinardi L., Seng H. Text Documents Plagiarism Detection using RabinKarpand

Jaro-Winkler Distance Algorithms, Indonesian J. of Electrical Engineering and Computer Science, 2017, v.5(2), P. 462-471.

13. Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.

14. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учебное пособие для прикладного бакалавриата / Д. Ю. Федоров. – 2-е изд., перераб. и доп. – Москва : Издательство Юрайт, 2019. – 161 с. – (Бакалавр. Прикладной курс). – ISBN 978-5-534-10971-9. – Текст: электронный // ЭБС Юрайт [сайт]. – URL: https://urait.ru/bcode/437489 (дата обращения: 30.04.2024).

15. Шелудько, В. М. Язык программирования высокого уровня Python. Функции, структуры данных, дополнительные модули: учебное пособие / В. М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 107 с. – ISBN 978-5-9275-2648-2. – Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. – URL: http://www.iprbookshop.ru/87530.html (дата обращения: 30.04.2024). – Режим доступа: для авторизир. пользователей.

REFERENCES

1. Levenshtein V.I. Binary codes with correction of dropouts, inserts and substitutions of characters // Dokl. USSR Academy of Sciences. 1965. 163 (4). C. 845-848.

2. Leshchenko A.V. Practical application of fuzzy search algorithms // Collection of scientific papers of NSTU. 2018. No.3-4(93). pp. 59-69.

3. Search for fuzzy matches: comparison of records by line spacing [Electronic resource]. URL: https://www.megaputer.com/ru/fuzzy-matching-comparing-records-with-stringdistance-measures / (date of access: 30.04.2024).

4. Wagner R. A., Fischer M. J. The string-to-string correction problem. J. ACM, v. 21(1), 1974. P. 168.

5. Damerau-Levenshtein Edit Distance Explained [Electronic resource]. URL: https://www.lemoda.net/text-fuzzy/damerau-levenshtein / (date of reference: 30.04.2024).

6. Jaro distance [Электронный ресурс]. URL: https://rosettacode.org/wiki/Jaro_distance(дата обращения: 30.04.2024).

7. Winkler W. E. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage, Proceedings of the Section on Survey Research Methods, American Statistical Association, 1990, P. 354-359.

8. Jaro-Winkler Similarity. Fuzzy string comparison [Electronic resource]. URL: http://xn---1-6kca8bgsjrjhe.xn --p1ai/public/1172479/ (accessed: 30.04.2024).

9. Code Spelunking: Jaro-Winkler String Comparison [Электронный ресурс]. URL: https://lingpipe-blog.com/2006/12/13/code-spelunking-jarowinkler-string-comparison/ (дата обращения: 30.04.2024).

10. Navarro G. A guided tour to approximate string matching, ACM Computing Surveys, 2001, v.33 (1), P. 31–88.

11. Winkler W. E. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage, Proceedings of the Section on Survey Research Methods, American Statistical Association, 1990, P. 354–359.

12. Brinardi L., Seng H. Text Documents Plagiarism Detection using RabinKarpand Jaro-Winkler Distance Algorithms, Indonesian J. of Electrical Engineering and Computer Science, 2017, v.5(2), P. 462-471.

13. Zlatopolsky D.M. Fundamentals of programming in Python. – M.: DMK Press, 2017. – 284 p.

14. Fedorov, D. Y. Programming in a high-level Python language : a textbook for applied bachelor's degree / D. Y. Fedorov. – 2nd ed., reprint. and additional – Moscow : Yurait Publishing House, 2019. – 161 p. – (Bachelor's degree. Applied course). – ISBN 978-5-534-10971-9. – Text: electronic // EBS Yurayt

[website]. – URL: https://urait.ru/bcode/437489 (date of access: 30.04.2024).

15. Sheludko, V. M. Python high-level programming language. Functions, data structures, additional modules: a textbook / V. M. Sheludko. – Rostov-on-Don, Taganrog: Southern Federal University Press, 2017. – 107 p. – ISBN 978-5-9275-2648-2. – Text: electronic // Electronic Library system IPR BOOKS: [website]. – URL: http://www.iprbookshop.ru/87530.html (date of access: 30.04.2024). – Access mode: for authorization users.