# Выбор алгоритма нечеткого поиска для PHP

В любой системе, где есть обработка ввода ключевых слов с клавиатуры, мы сталкиваемся с опечатками пользователей.

Хорошим тоном считается предоставить пользователю некоторое количество вариантов в стиле "Возможно, вы имели ввиду…", и затем перечислить некоторое количество похожих ключевых слов.

Алгоритмы нечеткого поиска по строкам– это та область, о которой вроде бы сказано достаточно много.

Но у нас несколько особенная ситуация, и я хочу рассказать об одном интересном опыте.

Предпосылки: существует проект, написанный на достаточно стандартной связке PHP + MySQL, в котором порой возникает необходимость помочь пользователю исправить опечатку в запросе или расширить поле запроса.

Поисковый движок Sphinx, который используется в нашем проекте, но к сожалению, не обладает "из коробки" возможностями нечеткого поиска в случае, если искажен корень искомого слова.  
Нечеткий поиск существует в Elasticsearch, а в PostgreSQL существует нечеткий полнотекстовый индекс, но в силу определенных ограничений сейчас речь пойдет о конкретном стеке.

Возник соблазн написать подсказчик на PHP, т.е. реализовать оптимальный алгоритм нечеткого поиска для формирования альтернативных запросов, если первый запрос ничего не вернул. И, конечно, это хотелось бы делать за наименьшее время и с наибольшей полнотой.

## Варианты

Условно их можно разбить на три подгруппы: алгоритмы с построением древовидного индекса, время которых приближается к сублинейному; фонетические алгоритмы, алгоритмы на конечных автоматах. Существуют также алгоритмы на нейросетях, и эта тема очень многообещающая, но в данной статье мы их не коснемся.   
 Я обнаружил, что популярные алгоритмы нечеткого поиска в PHP для русского языка реализованы достаточно слабо, и материалов в эту сторону почти не появлялось.

Вероятно, это потому, что PHP не считается подходящим языком для поиска по наборам данных, т. к. уже реализованы очень мощные алгоритмы нечеткого поиска в набирающих популярность СУБД наподобие Clickhouse.

Но если такая задача и возникнет, как, например у меня, то из коробки доступны только фонетические алгоритмы soundex-metaphone, которые не работают для русского языка, расстояние по Левенштейну (не Дамерау) и алгоритм Оливера (который медленнее алгоритма Левенштейна и несколько странно работает с Unicode)  
  
Я решил проверить несколько популярных алгоритмов и функций поиска

* Алгоритм хэширования по сигнатуре
* N-граммы
* BK-деревьев

И сравнить результаты по двум критериям: скорость и полнота.

В качестве образца полноты будет использоваться результат прямым перебором с метрикой Левенштейна (сложность O(n2)).

Как многие знают, расстояние Левенштейна между s1 и s2 – это количество элементарных операций (удаление; вставка; замена), необходимая для того, чтобы из строки s1 получить строку s2.

## Еще немного теории

### Алгоритм хэширования по сигнатуре.

Данный алгоритм прежде не был реализован на PHP и мне было особенно интересно попробовать его в данной задаче.

Идея заключается в следующем: алфавит, в котором происходит поиск, разбивается на M групп.

Теперь представим, что наш хэш – это некое число длиной M двоичных разрядов, в котором i-й разряд становится 1, если в слове наличествует символ алфавита из группы символов под номером i , и 0, если нет.

Методы разбиения здесь могут быть различные (скажем, по частоте, но для этого нужно знать частоты появления различных букв). Эффективно и быстро разбивать поровну,

т. е. номер группы для данного символа– это, грубо говоря, ord(symbol) — ord('а').

Если удалить или добавить одну букву, то изменится не более одного бита хэша. Может

не измениться ни один бит, если удаленная буква встречается в слове более одного раза, либо в слове есть буква из той же группы, что и удаленная буква. Если заменить один символ, то изменится не более двух битов хэша. В случае изменения двух битов, один бит обнуляется, а другой становится равным единице. [1]

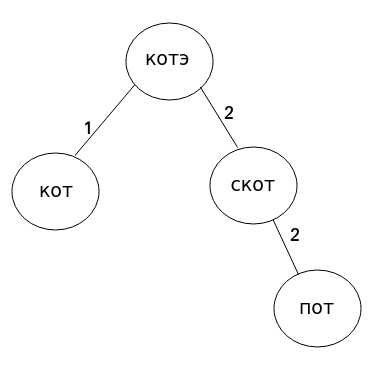
Поиск заключается в переборе хэшей с ошибками не более чем в MAX\_DISTANCE символов. Существуют и другие методики поиска.

Предполагается, что сложность поиска О(m2k), где m — длина хэша, а k — максимально допустимое расстояние в метрике.  
  
Существует множество разновидностей этого алгоритма, но, видимо, на PHP никто их не реализовывал.

### Алгоритм поиска по BK-деревьям

BK (Burkhard-Keller) -дерево– это связный ациклический граф, в котором ребра помечены расстоянием по выбранной метрике от i-го узла до j-го. Почему-то не удалось найти рабочую реализацию алгоритма на PHP.

Вначале выбирается случайное слово из словаря и назначается корневым элементом. Затем выбираем следующее слово из словаря, и пытаемся добавить его по следующему алгоритму: если расстояние от добавляемого элемента до корня равняется d, то помечаем это ребро d и добавляем в потомки к текущему узлу. Если уже существует узел с меткой d, то добавляем новое слово в потомки уже существующему.



На одном уровне не может быть двух узлов с одинаковой метрикой. Скажем, мы добавили "кот"с расстоянием 1, затем "скот" с расстоянием "2", и теперь добавляем "пот", тоже с расстоянием 2 к корневому узлу, то мы обязаны добавить его в тот, узел, который имеет такое же расстояние, т. е. в данном случае "скот".

Получается индекс, по которому проходим рекурсивно, отбирая среди узлов с метками между d- MAX\_DISTANCE до d+ MAX\_DISTANCE, где d-расстояние от искомого слова до текущего узла.

Сложность такого поиска O(Nr), где r < 1.

### Алгоритм N-грамм.

Давно известный алгоритм, почему-то не реализованный на PHP ранее (нашел только реализацию с поиском в MySQL)

Каждое слово разбирается на сэмплы длиной N. Сэмплы будут ключами в хэш-таблице, в которой по данному ключу будет храниться слово, содержащее N-грамму

Пример для N=3.



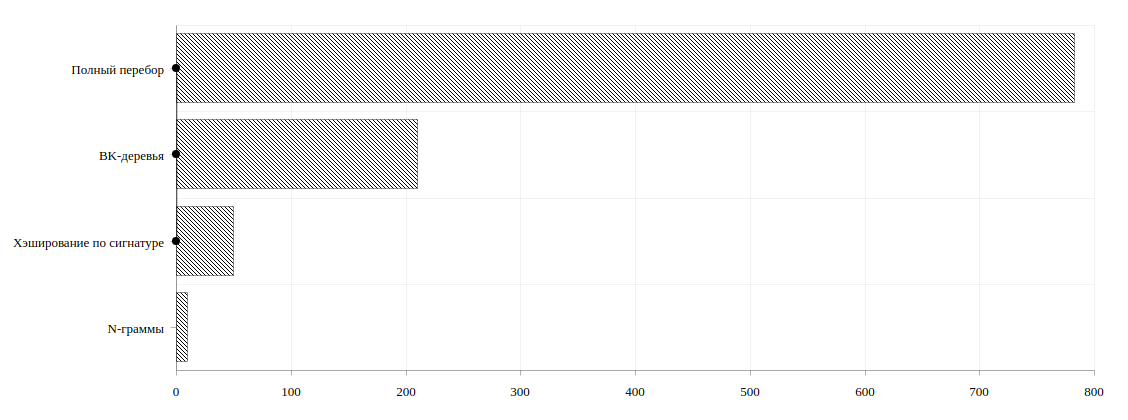
По данным ключам будет храниться слово "индекс". При поиске, искомое слово будет также разбито на N-граммы, каждая из которых послужит ключом для хэш таблицы. Те слова, метрика для которых меньше MAX\_DISTANCE, добавляются в результаты.  
Сложность алгоритма сублинейная.

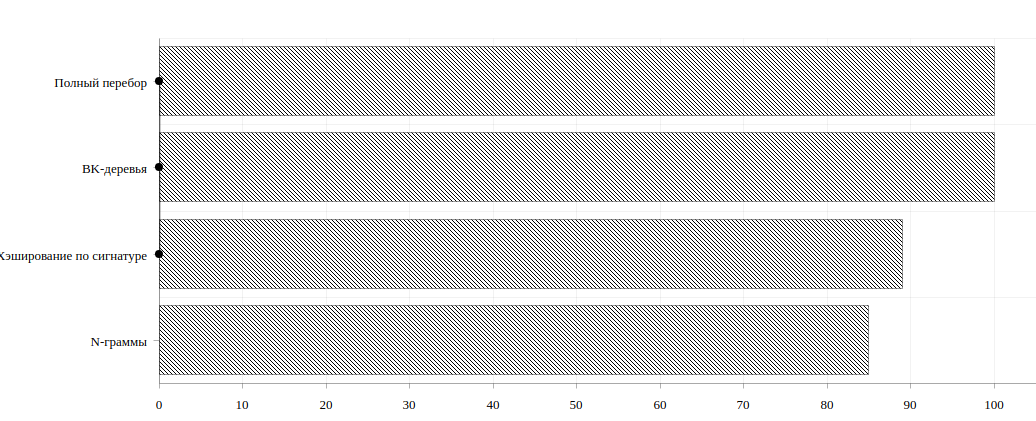
## Бенчмарки

Для тестирования был взят словарь А.А. Зализняка (93391 слово).

N-граммы тестировались при N=3.

Время поиска, мс.



Полнота, %

По соотношению полноты к скорости поиска, оптимальным способом будет выбор алгоритма N-грамм. Неплохой результат показало также хэширование по сигнатуре.  
Исходники можно посмотреть здесь

<https://github.com/YogSotot/PHPFuzzy>

## Что дальше

Из того, что я хотел рассмотреть, за пределами статьи остались различные вариации алгоритма хэширования по сигнатуре: поиск по TRIE-дереву, частотно-равномерная функция для построения хэша. Много полезной информации я почерпнул из [2]. Очень интересные алгоритмы нечеткого поиска подстрок были изобретены совсем недавно, например, [3]. Однако, не будем писать DOOM на микрокалькуляторе, для скромной задачи выбрать нечеткий поиск на PHP, я бы остановился на оптимизации уже рассмотренных алгоритмов.  
  
Литература  
1. Бойцов Л.М. Классификация и экспериментальное исследование современных алгоритмов нечеткого словарного поиска <http://rcdl.ru/doc/2004/paper27.pdf>

2. Сметанин Н. Нечеткий поиск в тексте и словаре <https://habr.com/ru/post/114997/>

3. Кутенин Д. Умные алгоритмы обработки строк в ClickHouse <https://habr.com/ru/company/yandex/blog/466183/>