Как работает автодополнение (поисковая подсказка, suggest)?

Автодополнение (поисковая подсказка, *suggest*) — это технология, которая в режиме реального времени предлагает пользователю возможные варианты завершения поискового запроса по мере ввода первых символов.

Требования к системе поиска подсказок:

1. Полнотекстовый поиск: Система должна находить подсказки, содержащие все слова из запроса пользователя, независимо от их порядка или местоположения в тексте подсказки. Пример: На запрос «смотреть» выдаются все подсказки, где встречается это слово.

2. Поиск по префиксам: Поиск должен работать на лету, в процессе набора, находя подсказки по начальным частям слов (префиксам). Пример: При вводе «смотр» должны находиться те же подсказки, что и для полного слова «смотреть».

3. Масштабируемость и скорость: Система должна обрабатывать десятки миллионов подсказок и возвращать результат за несколько миллисекунд, чтобы обеспечить мгновенную реакцию на действия пользователя.

Механизм автодополнения на основе префиксов

1. Лексикографическая организация данных

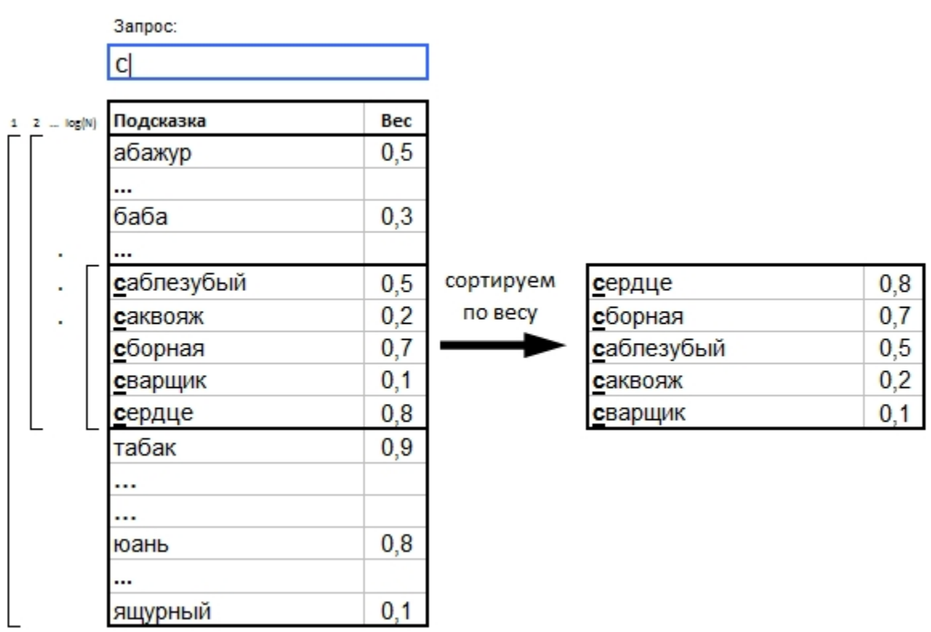
   Все подсказки предварительно сортируются в алфавитном порядке по своему тексту. Каждой подсказке присвоен вес, отражающий её популярность.

2. Поиск подходящего диапазона

При вводе пользователем префикса (начала запроса) с помощью бинарного поиска находится подмножество подсказок, текст которых начинаются с этого префикса.

3. Ранжирование и выдача результата

   Найденный блок подсказок пересортировывается по убыванию веса (популярности) и пользователю отображаются самые популярные подсказки из этого списка.



Для оптимизации используют Radix tree с весами в узлах дерева. Radix-tree (сжатое префиксное дерево) — это оптимизированная структура данных для хранения строк, где общие начала слов склеены в один узел, а ветвление происходит только при различиях. Помогает экономить память и ускоряет поиск за счёт объединения узлов (логарифмическое время поиска). [картинка]

Как работает?

1. Каждый узел знает максимум — максимальную популярность среди всех слов в своей ветви.

2. Последовательность поиска— сначала проверяем ветви с наибольшим максимумом.

3. Отсекаем бесперспективные — если максимум ветви ниже, чем худший из уже найденных результатов, ветвь не исследуем.

Полнотекстовое автодополнение

Алгоритм:

1. Токенизация

   Запрос пользователя и тексты подсказок разбиваются на отдельные слова (токены).

2. Поиск по условию

   Для заданного запроса система ищет в базе подмножество подсказок, удовлетворяющих условию: каждое слово из запроса должно совпадать с началом какого-либо слова в подсказке. Позиция этих слов внутри подсказки значения не имеет.

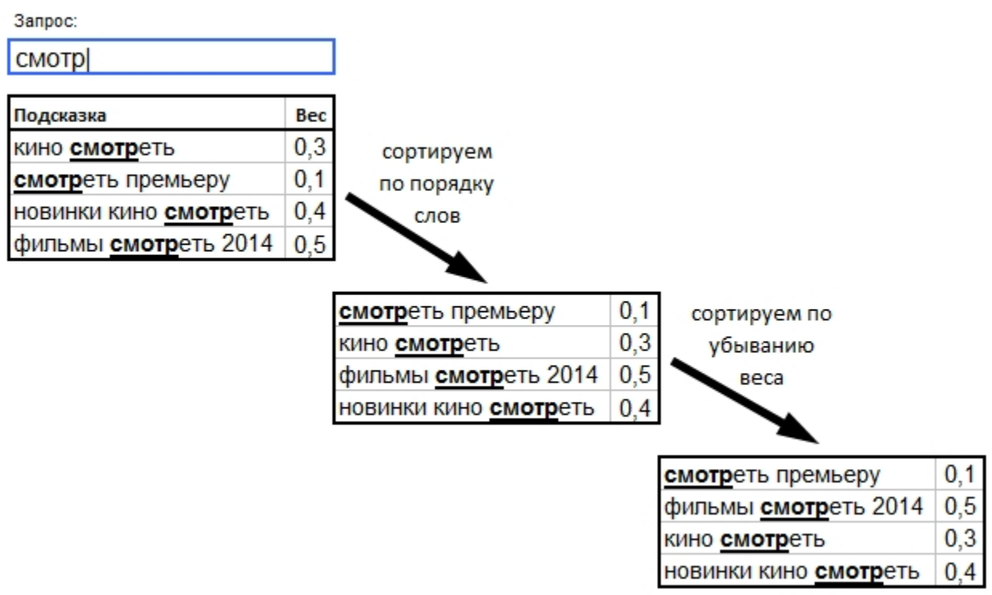
3. Ранжирование результатов

   Найденное подмножество сортируется по двум последовательным критериям:

   -По убыванию степени соответствия слов подсказки словам запроса. Учитывается близость и порядок найденных слов внутри подсказки.

   -По убыванию предустановленного веса (популярности) подсказки

4. Возврат результата пользователю



Для быстрого нахождения подсказок по словам запроса используются две взаимодополняющие структуры данных:

- прямой индекс

- обратный индекс

Прямой и обратный индексы

Прямой индекс — список документов, в котором можно найти этот документ по его id. Иными словами, прямой индекс — это массив строк (вектор документов), где id документа — это его индекс.

Обратный индекс — это словарь, составленный из всех уникальных слов, извлечённых из коллекции документов. Каждому такому слову сопоставлен упорядоченный список идентификаторов документов (posting list), содержащих данное слово.

По этим двум структурам данных достаточно легко найти все документы, удовлетворяющие пользовательскому запросу. Для этого нужно:

1. В обратном индексе: по словам из запроса найти списки id тех документов, где эти слова встречались. Получить пересечение этих списков — результирующий список id документов, где встречаются все слова из запроса. 2. В прямом индексе: по полученным id найти исходные документы и вернуть их пользователю.

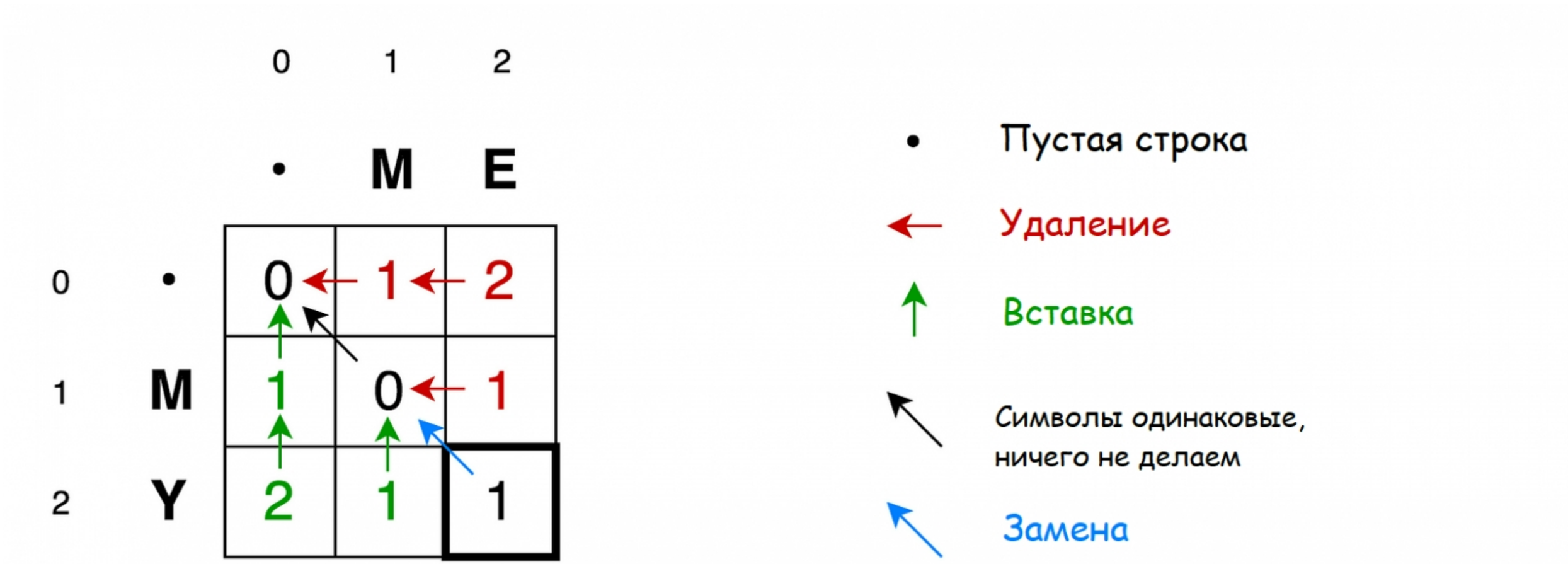
Описанные подходы обеспечивают быстрое автодополнение при точном вводе. Но что делать, если пользователь сделал опечатку или орфографическую ошибку? Для этого в базовых алгоритмах применяется нечеткий поиск.

Нечеткий поиск

Основные алгоритмы:

1. Расстояние Левенштейна

   Расстояние Левенштейна определяет какое минимальное количество односимвольных операций (вставки, удаления, замены) необходимо для превращения одной последовательности в другую.



   Как работает?

Если пользователь вводит запрос с ошибкой, то система проверяет похожие слова из базы и для каждого считает расстояние Левенштейна. Далее сортирует слова по возрастанию расстояния и возвращает пользователю результат с наименьшим расстоянием.

2. N-граммы

   Слово разбивается на подстроки фиксированной длины, ищутся слова с похожими фрагментами. Для запроса находятся все слова, у которых есть хоть один общий фрагмент и сортируются по количеству таких общих фрагментов. Пользователю возвращается слово с наибольшим количеством совпадений.

3. Поиск по синонимам и онтологиям

   Использование заранее созданных сетей связей между словами для понимания смысла, а не только буквенного состава.

Таким образом нечёткий поиск — это механизм, который позволяет системе находить релевантные варианты даже при наличии опечаток, орфографических ошибок или альтернативных написаний в запросе пользователя.

Подводя итог, автодополнение работает так: система в реальном времени ищет в своей базе подсказок варианты, наиболее похожие на уже введённую часть запроса. Для этого она использует префиксные деревья для мгновенного поиска по началу слов и инвертированные индексы для полнотекстового поиска по всем словам. Нечёткие алгоритмы подстраховывают, исправляя опечатки. Найденные варианты сортируются по популярности и релевантности, и топ лучших выводится пользователю — всё это за время между двумя нажатиями клавиш.

Источники: Поисковые подсказки изнутри Radix tree Более глубокое исследование: An Introduction to Information Retrieval