УДК 681.5:004.832.23
Замойский К. А.
Научный руководитель: ст. преподаватель Глущенко Т. А.

ЗАДАЧА НЕЧЕТКОГО ПОИСКА В ТЕКСТЕ И АЛГОРИТМЫ ЕЕ РЕШЕНИЯ

Вступление

Современные компьютерные системы поддерживают ограниченный набор средств для поиска по тексту. Обычно это только поиск полного вхождения искомой подстроки в строке из базы или индекса. Но для реализации полноценного и удобного поиска этого бывает недостаточно. Именно для этих целей применяют алгоритмы нечёткого поиска. Под нечетким поиском строки подразумевается такой поиск строки, когда поисковый шаблон или массив данных может подвергаться определенным искажениям. Примером применения нечеткого поиска строки может служить поиск подпоследовательностей ДНК после возможных мутаций или поиск текста, подверженного ошибкам набора и правописания.

Область применения

Задача анализа алгоритмов нечёткого поиска на сегодняшний момент актуальна, так как область применения данных алгоритмов невероятно велика и разнообразна. Начнём с распознавания рукописных символов, которое с массовым распространением устройств с сенсорным экраном активно используется для обеспечения удобства ввода. Введённый символ преобразуется в комбинацию цифр в зависимости от последовательности произведённых жестов, и полученная комбинация сравнивается со значениями, заранее известными для всех символов используемого алфавита, записанными в таблицу. Символ, для которого совпадение будет самым полным, и считается распознанным. Именно для определения полноты совпадения и используются алгоритмы нечёткого поиска, поскольку распознанные значения могут отличаться от заложенных в таблице для некого конкретного символа. Следующая область, где данные алгоритмы успешно применяются, это формы заполнения информации на сайтах и полноценные поисковые системы вроде Google или Yandex. Например, такие алгоритмы используются для функций наподобие «Возможно вы имели в виду ...» в тех же поисковых системах и для обнаружения опечаток в полях ввода программ. Так же алгоритм нечеткого поиска используется для извлечения данных для кластеризации серии спутниковых снимков Земли из космоса, поиск генетических последовательностей, компьютерное сжатие, индексации данных в поисковых системах, исправление ошибок в слове, сравнение текстовых файлов.

Анализ и понятие основных алгоритмов

Существует несколько подходов к решению задачи нечеткого поиска: поиск с использованием метрики Левенштейна, поиск с использованием алгоритмса Bitap, метод N –грамм, хеширование по сигнатуре, использование ВК-деревьев и ряд других методов. В данной статье рассмотрим некоторые из алгоритмов нечёткого поиска и дадим их сравнительный анализ. Программная реализация алгоритмов осуществлялась на языке C++, тестирование производилось с составлением собственного словаря английскх слов.

Рассмотрим несколько алгоритмов нечёткого поиска и отберём самые актуальные на данный момент для дальнейшего анализа. Начнём с двоичного алгоритма поиска подстроки (также bitap algorithm, shift-or algorithm) — алгоритм поиска подстроки, использующий тот факт, что в современных компьютерах битовый сдвиг и побитовое ИЛИ являются атомарными операциями. По сути, это примитивный алгоритм поиска с небольшой оптимизацией, благодаря которой, за одну операцию производится до 32 сравнений одновременно (или до 64, в зависимости от разрядности машины). Легко переделывается на приблизи-

тельный поиск. Вычислительная сложность — O(|n|·|h|) операций с крайне малой константой. Строка, которую мы ищем, имеет длину n, а строка, в которой ищем h. На предварительную обработку идёт O(|n|·|Σ|) операций и памяти, где Σ — алфавит. Если же длина шаблона поиска (в символах) не превышает длину машинного слова (в битах), сложность получается O(|h|) и O(|n|+|Σ|) соответственно. Шаблон найден, если найдена единица в последней строке этой матрицы.

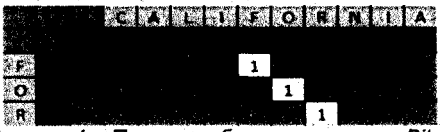

Рисунок 1 – Пример работы алгоритма Bitap

Алгоритмы нечеткого поиска характеризуются *метрикой* — функцией расстояния между двумя словами, позволяющей оценить степень их сходства в данном контексте. Наиболее известные метрики: – расстояния *Хемминга*, *Левенштейна* и *Дамерау-Левенштейна*.

Далее рассмотрим алгоритм Вагнера-Фишера, который позволяет для двух строк найти расстояние Левенштейна — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Пусть S_1 и S_2 — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) $d(S_1, S_2)$ можно посчитать по следующей рекуррентной формуле (1)

$$d(S_1, S_2)=D(M, N), \quad (1)$$

где

$$D(i,j)=\begin{cases} 0; i=0, j=0 \\ i; j=0, i>0 \\ j; i=0, j>0 \\ \min\begin{pmatrix} D(i,j-1)+1, \\ D(i-1,j)+1, \\ D(i-1,j-1)+m(S_1[i],S_2[j]) \end{pmatrix}; j>0, i>0 \end{cases} \quad (2)$$

Где *m(a,b)* равно нулю, если a=b и единице в противном случае; *min(a,b,c)* возвращает наименьший из аргументов. Данный алгоритм имеет ряд значительных преимуществ перед всеми описанными до этого, а именно: относительно невысокую сложность реализации, возможность качественного сравнения схожести более чем двух строк, несколько вариантов реализации, которые можно использовать в зависимости от конфигурации системы, универсальность для всевозможных алфавитов. К недостаткам же можно отнести, что при перестановке местами слов или их частей получаются сравнительно большие расстояния. Значения между совершенно разными короткими словами оказываются маленькими, в то время как между похожими и длинными строками — значительными.

		1	2	3	4	5	6	7	8	9
	1		2	3	4	5	6	7	8	9
	2	2		3	4	5	6	7	7	8
	3	3	3		4	5	6	7	8	8
	4	4	4	4		5	5	6	7	8
	5	5	5	5	5		6	5	6	7
	6	6	6	6	6	6		6	5	6
	7	7	7	7	7	7	7		6	

Рисунок 2 – Пример алгоритма Вагнера-Фишера

Следующий алгоритм — метод N-грамм. Был придуман довольно давно и является наиболее широко используемым, так как его реализация крайне проста, и он обеспечивает достаточно хорошую производительность. Алгоритм основывается на принципе: «Если слово А совпадает со словом Б с учетом нескольких ошибок, то с большой долей вероятности у них будет хотя бы одна общая подстрока длины N». Эти подстроки длины N и называются N-граммами. Во время индексации слово разбивается на такие N-граммы, а затем это слово попадает в списки для каждой из этих N-грамм. Во время поиска запрос также разбивается на N-граммы, и для каждой из них производится последовательный перебор списка слов, содержащих такую подстроку.

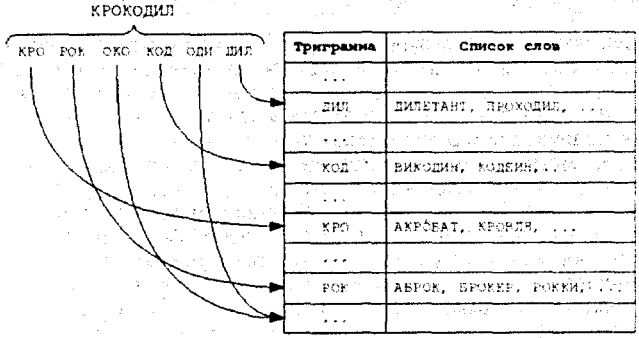

Рисунок 3 – Пример алгоритма N-грамм

Рассмотрим зависимость времени работы трех указанных алгоритмов от длины последовательности слов. Полученные результаты представлены на графике.

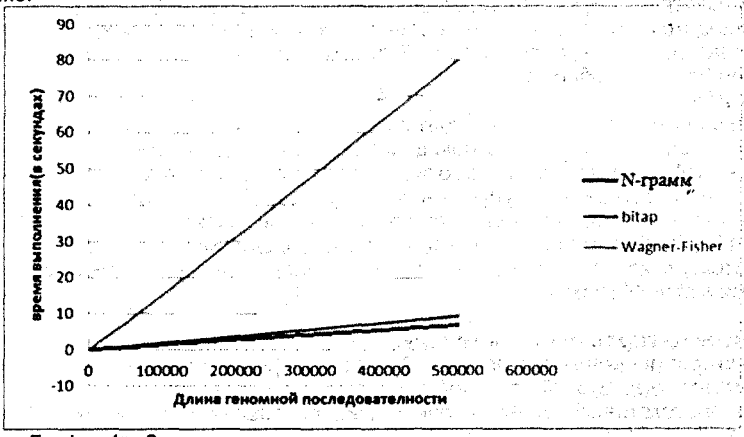

График 1 – Зависимость скорости выполнения алгоритма от длины последовательности слов

Приведем сравнительный анализ времени работы алгоритмов от кол-ва ошибок (0, 1, 2 соответственно). Полученные результаты сведем в таблицу.

Сравнительный анализ поиска слов (в миллисекундах)

Пример слов: interstellar (1), temperature (2), regeneration (3).

Таблица 1 – сравнение скорости выполнения алгоритмов для заданных слов с учетом их правильного (неправильного) ввода

Название алгоритма	0 Ошибок			1 Ошибка			2 Ошибки		
	1-слово	2-слово	3-слово	1-слово	2-слово	3-слово	1-слово	2-слово	3-слово
N-грамм	5	8	5	5	5	4	5	7	5
Bitap	9	8	5	—			—		
Вагнера-Фишера	14	12	14	13	16	14	12	15	15

Анализируя полученные результаты, я понял, что каждый из них имеет свои преимущества относительно скорости, памяти, реализации и т. д. Что касается алгоритма Bitap , то сложность данного алгоритма составляет O(mn), что в принципе не так мало. Но у алгоритма есть и приятные стороны. Во-первых, он очень прост в реализации и в понимании того, как он работает. Во-вторых, так как каждый столбец матрицы вычисляется на основании предыдущего, то нам нет необходимости хранить в памяти всю матрицу, а достаточно держать только два столбика. На практике же алгоритм хорошо справляется при поиске небольших образцов, таких как, например, английское слово. Алгоритм Вагнера-Фишера требует O(mn) операций и такую же память. Последнее может быть неприятным: так, для сравнения файлов длиной в 105 строк потребуется около 40 гигабайт памяти. Что касается N-грамм тут все зависит от того, какое N мы возьмем. Выбор большего значения N ведет к ограничению на минимальную длину слова, при которой уже возможно обнаружение ошибок. Чем меньше длина слова и чем больше в нем ошибок, тем выше шанс того, что оно не попадет в соответствующие N-граммам запроса списки и не будет присутствовать в результате. Между тем, метод N-грамм оставляет полный простор для использования собственных метрик с произвольными свойствами и сложностью, но за это приходится платить — при его использовании остается необходимость в последовательном переборе около 15% словаря, что достаточно много для словарей большого объема.

Заключение

Большинство алгоритмов нечеткого поиска с индексацией не являются истинно сублинейными (т. е. имеющими асимптотическое время работы O(log n) или ниже), и их скорость работы обычно напрямую зависит от N. Тем не менее, множественные улучшения и доработки позволяют добиться достаточного малого времени работы даже при весьма больших объемах словарей. Данная работа показала, какие задачи может решать нечеткий поиск. Исходя из полученных результатов, среди данных алгоритмов лучшим является метод N-грамм.

Список цитированных источников

1. Алгоритмы нечеткого поиска // Хабрахабр [Электронный ресурс]. – 2018. – Режим доступа: https://habr.com/post/114997/.

2. Полнотекстовый нечеткий поиск с использованием алгоритма Вагнера-Фишера // Хабрахабр [Электронный ресурс]. – 2018. – Режим доступа: https://habr.com/post/279585/.

3. N-грамма // Википедия [Электронный ресурс]. – 2018. – Режим доступа: https://ru.wikipedia.org/wiki/N-грамма.

4. Расстояние Левенштейна // Википедия [Электронный ресурс]. – 2018. – Режим доступа: https://ru.wikipedia.org/wiki/Расстояние_Левенштейна