СИСТЕМЫ УПРАВЛЕНИЯ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Основан в 1995 г.

2020
№ 2(80)

2020

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций **ПИ N ФС77-66093 от 10 июня 2016 г. (**первичная регистрация **от 20 мая 2003 г.)**

ISSN 1729-5068

Журнал выходит четыре раза в год

СИСТЕМЫ УПРАВЛЕНИЯ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Дизайн обложки Т.А.Бурковская

Адрес учредителя и редакции:
394026 Воронеж, Московский проспект, дом 14

Телефон: (473)2437718
E-mail: csit@bk.ru
http://www.sbook.ru/csit/

16+

Учредитель: ФГБОУ ВО «Воронежский государственный технический университет»
Издатель: ООО Издательство «Научная книга» http://www.sbook.ru
Адрес издателя: 394077 Воронеж, 60-й Армии дом 25-120

Цена свободная.
Отпечатано с готового оригинал-макета в ООО «Цифровая полиграфия»
394036, г.Воронеж, ул.Ф.Энгельса, 52, тел.: (473)261-03-61

Подп. в печать 11.05.2020. Дата выхода в свет 11.06.2020. Заказ 000. Тираж 500. Усл. печ. л. 10,2.

Содержание

tific Press, Portland, OR, 2012. - 326 p.

17. Агиева М.Т., Бабичева Ю.В., Окулист Н.М., Угольницкий Г.А. Имитационное моделирование управления мнениями в маркетинге // Системы управления и информационные технологии, 2019, 4(78), 61-65.

18. Угольницкий Г.А., Усов А.Б. Алгоритмы решения дифференциальных моделей иерархических систем управления // Автоматика и телемеханика, 2016, 5, 148-158.

УДК 004.651

Андрианов И.А., Григорьева А.Н.

МОДЕРНИЗАЦИЯ ИНДЕКСА ДЛЯ ПОИСКА ПО РЕГУЛЯРНЫМ ВЫРАЖЕНИЯМ *

Вологодский государственный университет

В статье описывается модернизация индекса для поиска по регулярным выражениям на основе подстрок переменной длины - мультиграмм. В результате удалось существенно снизить время выполнения широкого ряда запросов. Также в статье представлен модифицированный алгоритм обновления индекса при вставке новых документов в базу данных.

Введение

С помощью регулярных выражений можно формулировать довольно гибкие запросы для поиска в текстовых данных. Например, так может выглядеть выражение для поиска ссылок на mp3-файлы “<a href=("|’)?.*\.mp3("|’)?> ” (для примера оно несколько упрощено).

В стандарте языка SQL (начиная, как минимум, с SQL-92) имеется оператор LIKE. По сути, LIKE-шаблоны представляют собой подмножество регулярных выражений, записанных с немного отличающимся синтаксисом. Некоторые СУБД поддерживают и полноценные регулярные выражения - например, в PostgreSQL для этого используется оператор “~”.

Несмотря на довольно широкое применение регулярных выражений, производители СУБД не спешат разрабатывать специализированные виды индексов для ускорения таких запросов. Традиционные же индексы (на базе B+-деревьев, хеш-таблиц или битовых карт) для большинства регулярных выражений непригодны. В результате выполнение большинства таких запросов требует полного сканирования всей таблицы.

Анализ работ в данной области показывает, что можно выделить несколько подходов к построению индексов для регулярных выражений. Первый способ предполагает использование суффиксных боров, деревьев Patricia [1], суффиксных деревьев [2, 3, 4]. Существенным недостатком таких структур является высокое потребление памяти, что затрудняет их использование для больших баз данных. Проблема усугубляется плохой пространственной локальностью: например, при выполнении поиска в суффиксном дереве требуется перемещаться по различным его узлам, что вызывает многократное чтение разных дисковых страниц.

Возможный способ преодоления данного недостатка − организовать размещение дерева во внешней памяти так, чтобы уменьшить количество обращений к диску при построении дерева и при выполнении поиска. Наиболее известными среди таких методов являются TDD, Trellis, B^{2}ST и DiGeST, их обзор можно найти в [5]. Альтернативный способ заключается в разработке структуры данных, каким-либо образом сочетающей свойства структур данных для оперативной и для внешней памяти. Широко известным примером является строковое B-дерево (String B-tree) [6], которые является комбинацией B-дерева и Patricia-дерева.

Ещё один подход заключается в использовании в качестве ключей индекса q-грамм - всевозможных подстрок входных данных фиксированной длины q [7]. С каждой q-граммой связывается список вхождений - перечень документов, в которых эта грамма встречается.

Важный частный случай возникает при q=3 - такой индекс называется триграммным, он легко реализуется и зачастую даёт неплохие результаты. Примером является расширение pg_trgm для СУБД PostgreSQL. Недостаток триграммного индекса заключается в том, что далеко не всегда он ускоряет выполнение запросов, даже если запрос имеет хорошую селективность. Для примера рассмотрим поиск подстроки “fort” в базе с исходными текстами на языке C. Заметного ускорения, вероятно, получено не будет, поскольку триграмма “for” содержится почти в каждом документе, триграмма “ort” встречается тоже довольно часто (как часть слова “short”). Увеличение же длины грамм приведёт к многократному увеличению размера индекса, что затрудняет его использование в больших базах данных.

Интересный подход к построению индекса предложен в работе [8]. В ней предлагается в качестве индексируемых элементов использо-

*Работа частично поддержана грантом РФФИ 18-47-350001 р_а

вать подстроки переменной длины - так называемые мультиграммы. При этом, чтобы достичь низких требований к памяти, используются следующие два правила. Во-первых, в индекс включаются лишь только «полезные» мультиграммы - то есть встречающиеся лишь в небольшой части входных документов. Во-вторых, никакая полезная мультиграмма не должна быть подстрокой другой - в этом случае более длинная мультиграмма в индекс не включается. В дальнейшем данная идея была развита в работе [9] - в ней для выбора ключей индекса предлагается дополнительно учитывать, какие поисковые запросы встречаются наиболее часто, чтобы оптимизировать индекс в первую очередь для ускорения этих и похожих запросов. В статье [10] для этой цели используется метод линейного программирования.

В наших работах [11, 12] было предложено расширить множество ключей в ядре такого индекса, а также изменить определение их селективности. В результате была достигнута возможность обновления индекса без его полной перестройки, а также повысилась эффективность поиска для определённого класса запросов.

В данной статье мы предлагаем расширить и дополнить описанные в [11, 12] модификации структуры и алгоритмов работы с индексом с целью дальнейшего повышения его производительности и для расширения класса поддерживаемых запросов.

Структура и алгоритм построения индекса

В [8] было предложено сделать ключами индекса лишь мультиграммы, которые наиболее «полезны» для поиска. Полезность граммы *x* определяется её селективностью:

$$Sel(x) = M(x)/N \quad (1)$$

Здесь N - количество (записей) в таблице базы данных, M(x) - число записей, содержащих грамму x в качестве подстроки. При этом грамма x будет считаться полезной, если величина Sel(x) не превосходит определённой константы c. Константа c подбирается с учётом решаемой задачи, в большинстве случаев величина 0.01 будет подходящим значением.

Чтобы ещё более снизить размер индекса, используется следующее правило: никакой ключ в индексе не должен являться подстрокой другого ключа. Если грамма a является подстрокой граммы b, то в индекс включается только грамма a (как более короткая). Кроме того, максимальную длину мультиграмм, включаемых в индекс, ограничивают некоторой константой L (в [8] использовалось L=10).

Для полученного множества ключей строятся списки вхождений так же, как это делается в обычном инвертированном индексе. В [8] доказано, что суммарный размер списков вхождений не превышает суммарного размера входных данных. Важно отметить, что ядро индекса (то есть набор его ключей) оказывается настолько компактным, что помещается целиком в оперативной памяти даже для входных данных объёмом в десятки гигабайт.

Алгоритм построения индекса состоит из двух стадий [8]. На первой стадии строится префиксно-свободное множество полезных грамм (в котором никакая грамма не является префиксом никакой другой). Псевдокод алгоритма представлен на рис. 1.

```
k = 1, expand = {""}
while ((expand не пусто) and ( k ≤ L))
  k-grams := все k-граммы в базе данных,
             у которых (k-1)-префикс ∈ expand
  expand := {}
  for each gram x in k-grams
    if sel(x) ≤ c then // проверка селективности
      insert(x, index) // грамма полезна
    else
      expand := expand ∪ {x}
  k := k + 1
```

Рис. 1. Псевдокод алгоритма построения индекса

Поясним приведённый алгоритм. В нём происходит максимум L проходов по исходным данным. На каждом из них бесполезные мультиграммы, имеющие длину k, увеличиваются на один символ и разделяются в соответствии с селективностью на два класса - бесполезные и полезные. Полезные мультиграммы отправляются в результирующее множество, а для бесполезных грамм алгоритм повторяется.

На следующем этапе полученное префиксно-свободное множество конвертируется в пресуффиксно-свободное. С этой целью все элементы множества переворачиваются и упорядочиваются по возрастанию. Теперь все лишние граммы можно отбросить за один проход, а оставшиеся граммы переворачиваются ещё раз. В окончательном множестве никакая мультиграмма не будет являться подстрокой никакой другой.

Рассмотрим пример. Пусть в базе данных содержатся три документа - “aba”, “caba” и “dab”, при этом требуемый уровень селективности c=2/3. После первой стадии алгоритма получится множество {“aba”, ”ba”, “c”, “d”}.

Заметим, что грамма “ba” является суффиксом граммы “aba”, то есть на второй стадии алгоритма элемент “aba” будет устранён. Окончательное множество ключей индекса равняется {“ba”, “c”, “d”}. На последнем этапе остаётся построить для них списки вхождений.

Модификации структуры индекса и алгоритма его построения

Один из недостатков вышеописанного индекса состоит в том, что он не позволяет ускорять большой класс запросов, возвращающих ноль записей.

Рассмотрим пример. Пусть в базе данных содержатся строки “abc”, “abd”, “abe”, “abx”, “aby”, а порог селективности c=0.2. При этом множество ключей индекса будет равным {“c”, “d”, “e”, “x”, “y”}. Теперь предположим, что в базе содержатся строки “ac”, “ad”, “ae”, “bx”, “by”. Множество ключей индекса получится точно таким же.

Предположим, мы ищем подстроку “ab”. В первом случае она встречается во всех записях, во втором - не встречается вообще. Однако, индекс не способен отличить эти два случая друг от друга. В результате потребуется выполнять полное сканирование всех данных даже если искомой подстроки в базе нет.

Чтобы сделать индекс применимым и для поиска подстрок, отсутствующих в базе, мы предлагаем расширить множество ключей индекса, добавив в него некоторое подмножество бесполезных мультиграмм. Оно будет определяться следующим образом. Пусть U - множество всех бесполезных грамм длиной не более L. Добавим в индекс лишь такие элементы $a \in U$, что не существует такого $b \in U$, что a является префиксом b. Простыми словами, мы добавим в индекс бесполезные граммы максимальной длины, не превышающей L.

Отметим, что для таких бесполезных грамм не нужно строить списки вхождений - достаточно лишь хранить количество документов, в которых они встречаются.

Интересно, что предложенное изменение состава ключей индекса потребует внесения лишь незначительных изменений в приведённый выше алгоритм. Пусть $x \in expand$ - очередная грамма длины (k-1), расширяемая до длины k. Выполняя расширение, будем дополнительно запоминать, получилась ли в результате расширения хотя бы одна бесполезная грамма. Если да, то грамма x не является максимальной и в ответ не включается, в противном случае x добавляется в индекс.

Важный вопрос - насколько сильно при этом увеличится размер индекса. Пусть N - количество входных документов, D - их суммарный размер, c - установленный порог селективности. Утверждение: количество бесполезных грамм, дополнительно добавленных в индекс, не превышает следующей величины:

$$Useless \le D/(N \cdot c+1) \qquad (2)$$

Доказательство. Пусть D - суммарный размер всех входных данных. Если в какой-то позиции какого-то входного документа начинается несколько бесполезных грамм, то в ответ попадёт только самая длинная из них. Получается, что количество таких грамм не превышает D. Но, поскольку граммы бесполезные, то каждая из них встречается не менее чем в N·c различных позициях входных данных, что и приводит нас к неравенству (2).

Из данного неравенства следует, что количество дополнительно добавленных в индекс грамм не может превышать размер входных данных. На практике же величина $N \cdot c$ практически всегда намного больше единицы - то есть размер индекса возрастает незначительно (экспериментальные оценки приведены в конце данной статьи).

Дополнительно отметим, что приведённый в предыдущем разделе алгоритм можно упростить, полностью исключив из него вторую фазу. Пусть мы получили полезную грамму x длины k. Вместо того, чтобы добавлять её в индекс, проверим, а не содержится ли в уже в индексе суффикс этой граммы длины (k-1). Если такой суффикс найдётся, то грамма x не является минимально полезной и должна быть отброшена. Несложно видеть, что в итоге будет получен тот же самый результат.

Алгоритм обновления индекса при вставке новых записей

Предложенное нами в предыдущем пункте изменение в структуру индекса имеет ещё одно важное применение - оно позволяет сделать индекс обновляемым. В результате при вставке новых данных в базу становится возможным обновлять индекс многократно быстрее по сравнению с его полной перестройкой.

Чтобы сделать индекс обновляемым, вначале внесём изменения в определение полезности грамм. Будем считать грамму x полезной, если она встречается не более чем в T входных документов. Значение порога T подбирается исходя из конкретной предметной области. Данное изменение позволяет избежать ситуации, когда сразу большое количество бесполезных грамм превращаются в полезные после добавления всего одного документа в базу данных.

Алгоритм обновления индекса напоминает алгоритм его построения с нуля, но имеет несколько особенностей, которые следует учитывать. Пусть useful - множество полезных грамм, входящих в индекс, useless - множество бесполезных грамм в индексе. Взяв очередную грамму x из добавляемого документа, вначале проверим, не содержится ли её префикс в useless. Если содержится, то грамма является бесполезной и оставляется для дальнейшего расширения.

В противном случае проверим, найдётся ли хотя бы одна грамма с таким префиксом во множестве useful, и обновим счётчик документов для граммы x. Если грамма x осталась полезной, то добавляем (или обновляем) её в useful. Если грамма бесполезна, но не содержится в *useful*, то оставляем её для дальнейшего расширения.

Наиболее интересный момент возникает, когда ранее полезная грамма становится бесполезной. При этом потребуется взять её список вхождений и прочитать с диска все документы, в которых эта грамма встречается. Аналитическая оценка того, насколько часто может возникать такая ситуация, достаточно сложна и составляет одно из направлений дальнейшего исследования.

Однако, вычислительные эксперименты для разных типов входных данных показывают, что на практике при вставке документа превращение полезных грамм в бесполезные случается достаточно редко и затрагивают лишь тысячные доли процента от всех полезных грамм в рассмотренном индексе.

Вычислительные эксперименты и анализ их результатов

Для проведения экспериментов мы использовали три вида входных данных - набор литературных текстов на естественном языке, набор исходных программных файлов на языке C и случайно сгенерированные тексты с равномерным распределением символов.

Объём данных каждого вида был взят сравнительно небольшой - около 0.3 Гб. При этом время построения индекса на ноутбуке авторов составляет лишь около пяти минут, что удобно для проверки различных версий кода.

Стоит отметить, что нами тестировались программные прототипы, не содержащие какой-либо существенной оптимизации. Можно предположить, что хорошо оптимизированный программный код позволит получить значительно более высокую производительность.

В первом эксперименте мы замерили количество и размер полезных и бесполезных мультиграмм в индексе, а также суммарный размер списков вхождений. Каждый набор входных данных содержал примерно 20000 документов, использовались параметры: T =20, L = 10. Результаты представлены в таб. 1.

Эксперимент подтвердил, что количество и суммарный размер добавленных в индекс бесполезных мультиграмм оказывается в несколько раз меньше, чем полезных, т.е. наше изменение индекса незначительно увеличивает его размер. Важно также отметить, что суммарный размер всех ключей индекса во всех случаях оказался значительно меньше, чем суммарный размер входных данных, что во многих случаях позволяет кэшировать всё ядро индекса в оперативной памяти.

Во втором эксперименте мы замерили среднее время обновления индекса при добавлении одного документа, а также среднюю частоту возникновения ситуации, когда полезная мультиграмма превращается в бесполезную. Результаты представлены в табл. 2.

Таблица 1

Размер индекса для разных видов данных

	Литературные тексты	Исходный код	Случайные данные
К-во полезных мультиграмм	8645082	6948928	14348907
Суммарная длина полезных мультиграмм	72258144	49806275	71744535
К-во бесполезных мультиграмм	3086620	1606616	531441
Суммарная длина бесполезных мультиграмм	28228015	13324119	2125764
Суммарный размер списков вхождений	60077351	43284672	299732452
Время построения, с	333	216	145

Таблица 2

Время обновления индекса при вставке записи

	Литературные тексты	Исходный код	Случайные данные
Время обновления индекса, с	2.9	2	1.9
К-во полезных мультиграмм, ставших бесполезными	246	173	40

Из табл. 2 можно видеть, что время обновления индекса оказалось примерно на два порядка меньше, чем его полная перестройка. Такой результат подтверждает возможность использовать предложенный способ индексирования не только для статических, но и для динамических наборов данных.

Заключение

В статье мы предложили внести изменения в структуру и алгоритм построения мультиграммного индекса для ускорения поиска по регулярным выражениям. Это позволило многократно повысить скорость выполнения широкого ряда запросов - особенно тех, где ищется отсутствующая в базе данных информация. При этом размер индекса увеличился незначительно.

Также в статье представлен алгоритм обновления индекса при вставке новых документов в базу данных. Эксперименты с ним показали примерно на два порядка более высокую скорость обновления индекса по сравнению с его

полной перестройкой.

Список использованных источников

1. Baeza-Yates R., Gonnet G. Fast text searching for regular expressions or automaton searching on tries// Journal of the ACM, 43(6). 1996. P. 915-936. DOI: 10.1145/235809.235810.

2. Hunt E., Atkinson M., Irving R. Database indexing for large dna and protein sequence collections// VLDB, 11. 2002. P. 256-271. DOI: 10.1007/s007780200064.

3. Zuopeng L., Kongfa H., Ning Y., Yisheng D. An efficient index structure for XML based on generalized suffix tree// Information Systems, 32, Issue 2. 2007. P. 283-294. DOI: 10.1016/j.is.2005.10.001.

4. Андрианов И.А., Свиньин С.Ф. Применение равномерно разреженных суффиксных деревьев для задач обработки строк// Труды СПИИРАН, 3 (34). - 2014. - С. 247-260.

5. Barsky M., Stege U., Thomo A. Full-Text (Substring) Indexes in External Memory// Synthesis Lectures on Data Management, 3, No.7. 2011. P. 1-92. DOI: 10.2200/S00396ED1V01Y201111DTM022.

6. Ferragina P, Grossi R. The String B-Tree: A New Data Structure for String Search in External Memory and its Applications// Journal of the ACM, 46, Issue 2. 1999. P. 236-280. DOI:10.1145/301970.301973.

7. Navarro G., Sutinen E., Tarhio J. Indexing text with approximate q-grams. CPM2000// Lecture Notes in Computer Science. 2000. P. 350-363. DOI:10.1007/3-540-45123-4_29.

8. Cho J., Rajagopalan S. A fast regular expression indexing engine// Proc. 18th Int. Conf. on Data Engineering. 2002. P. 419-430. DOI:10.1109/ICDE.2002.994755

9. Hore B., Hacigumus H., Lyer B., Mehrotra S. Indexing text data under space constraints// CIKM '04: Proc. of the thirteenth ACM int. conf. on Information and knowledge management. 2004. P. 198-207. DOI: 10.1145/1031171.1031212

10. Tsang D., Chawla S. A robust index for regular expression queries// CIKM'11: Proc. of the 20th ACM Conf. on Information and Knowledge Management. 2011. P. 2365-2368. DOI:10.1145/2063576.2063968.

11. Андрианов И.А., Григорьева А.Н. Разработка обновляемого индекса для поиска по регулярным выражениям и сходству // Информационные системы и технологии, №6 (110), 2018. - С. 74-82.

12. Андрианов И.А., Григорьева А.Н. Эффективный поиск плагиата в программном коде для системы дистанционного практикума по программированию// Тр. Междунар. Нпк «ИНФОРИНО-2016» - М.: ИД МЭИ, 2016. - С. 485-488.

УДК 004.056.053

Бородащенко А.Ю., Жусов Д.Л., Козленко А.В., Макеев С.М.

УПРАВЛЕНИЕ ИНФОРМАЦИОННЫМИ РЕСУРСАМИ ЗАЩИЩЕННЫХ WEB-ПОРТАЛОВ

Академия Федеральной службы охраны Российской Федерации

В работе представлены результаты моделирования задачи управления информационными ресурсами защищенных web-порталов. Описаны способы оценки защищенности и контроля целостности информационных web-ресурсов.

Введение

Нарушение функционирования компьютерных сетей (КС) большинства организаций вследствие несанкционированного доступа (НСД) к обрабатываемой в них информации влечет за собой значительный ущерб. Достижение научно-обоснованного уровня защищенности информации от НСД обуславливает необходимость проведения его оценки и контроля защищенности информационных ресурсов (ИР) на всех этапах жизненного цикла КС при различной степени полноты и достоверности имеющейся информации. В настоящее время в КС активно применяется технология web-порталов, представляющая следующие преимущества [1]:

- предоставление пользователям единой точки персонифицированного доступа к информационным ресурсам;
- интеграция разнородных информационно-аналитических и информационно-справочных систем;
- унификация подходов к использованию большого количества существующих и разрабатываемых перспективных информационных систем;
- масштабируемость системы (увеличение количества пользователей, подключение новых ИР, создание дополнительных портальных сервисов и т.д.).

Таким образом, представляется актуальной задача управления ИР web-порталов, включающей в себя этап оценки их защищенности и контроль защищенности по результатам оценки.

Оценка защищенности информационных ресурсов ведомственных web-порталов

Согласно требованиям нормативных документов (руководящих документов Федеральной службы по техническому и экспортному контролю РФ, ГОСТ Р ИСО/МЭК 15408-2008, ГОСТ Р ИСО/МЭК 17799-2005 и т.д.) преобладают качественные подходы к оценке защи-