Индексирование путей в слабоструктурированных данных *

Б.С. Хвостиченко, Б.А. Новиков
Санкт-Петербургский государственный университет

Аннотация

Рассматривается задача построения индексной структуры для слабоструктурированных данных, пригодная для выполнения запросов, содержащих регулярные выражения для путей.

Предлагается метод индексирования, основанный на использовании B^*-дерева со сжатием ключей для хранения путей, представленных строками.

1 Введение — слабоструктурированные данные

Требования, предъявляемые к современным Web-приложениям, зачастую выражаются в терминах управления слабоструктурированными данными (обычно представляемыми с использованием XML).

Термин "слабоструктурированные данные" используется с середины 90-х годов для обозначения данных, структура которых задается менее жестко, чем в более традиционных реляционных базах данных, характеризующихся высокой степенью полноты и регулярности.

Такого рода информация может характеризоваться тем, что некоторые данные могут отсутствовать, одинаковые же могут быть представлены в различном виде. Более того, структура данных может быть вообще неизвестна пользователю. Примером таких данных может служить классический B2B каталог продуктов, где данные от различных поставщиков должны быть объединены таким образом, чтобы покупатель мог осуществлять поиск по ним.

В этой работе предполагается, что слабоструктурированные данные представляются как направленный граф, вершины которого — это объекты, соединенные помеченными дугами-отношениями. Такая модель данных носит название OEM и более подробно рассматривается в [2]. Данные в модели XML [9] тоже могут рассматриваться, как ориентированный граф, в котором не может быть контуров (т.е. граф на самом деле является деревом). Для хранения слабоструктурированных данных можно как представлять граф в уже существующих RDBMS или OO DBMS, так и использовать специально разработанные системы (например, Lore [5]).

Одной из особенностей слабоструктурированных данных является то, что при поиске могут задаваться запросы, специфицирующие пути к объектам, то есть могут выбираться данные, достижимые по определенному пути из корня (рис. 1). Для обеспечения поиска в неизвестной структуре данных необходимо допустить частичную спецификацию пути. Для представления частично специфицированных путей естественно применение регулярных выражений в запросе пути (рис. 2).

*Работа частично поддержана РФФИ, грант 01-01-00935

```
SELECT x.address.street
    FROM cafes x
        WHERE x.cuisine = "traditional"
```

Рис. 1: Пример поиска по пути

```
SELECT x(.address)?.street
    FROM cafes x
        WHERE x.*.cuisine = "traditional"
```

Рис. 2: Пример регулярного выражения пути в запросе

Известно, однако, что обход дерева имеет экспоненциальную временую сложность по глубине дерева, поэтому поиск по пути достаточно трудоемок. Решение подобной проблемы — построение индексной структуры, использование которой бы сокращало время выполнения запроса.

В работе описывается предлагаемая индексная структура, а также механизмы индексирования и поиска ключей.

2 Другие подходы

За последние три года вопросу индексирования слабоструктурированных данных уделялось достаточно большое внимание исследователей, но практически все предложенные структуры специально не предназначены для поиска путей, задаваемых регулярными выражениями.

Одна из практических попыток была предпринята группой разработчиков проекта Lore [5], которые предложили четыре индексные структуры [6]:

Vindex Индекс для значений атомарных объектов (с преобразованием типов).

Tindex Индекс для атомарных строк.

Lindex Хранит указатели на родителей объекта.

Pindex Предоставляет возможность быстрого поиска объекта по заданному пути.

Еще одна структура, используемая в системе Lore для ускорения запросов, называется *Dataguide* [4]. Это сжатое описание схемы данных, но к сожалению оно никак не учитывает реальные отношения родитель-потомок между узлами.

Другая, более теоретическая работа проделана Suciu и Milo в [7], где они разработали общую модель индексирования слабоструктурированных данных. Их *T-index* учитывает возможность запроса регулярного выражения пути, но к сожалению он достаточно сложен в построении. К тому же, этот метод предложен для индексирования запросов, отвечающих конкретному образцу (*T-template*), т.е. для каждого образца необходимо строить отдельный индекс. Авторы дают более детальное описание и оценки сложности для двух простейших случаев: *1-index* и *2-index*. В первом случае опять же не учитываются реальные отношения родитель-потомок между объектами, а *2-index* уже достаточно сложен в построении.

Еще одна индексная структура, названная ToXin [8], была использована в системе ToX. В ней разработчики объединили *Vindex* — для индексирования значений, *Dataguide* — схему данных и *instance function* — для каждой помеченной дуги в *Dataguide* эта функция возвращает пары <**родитель, потомок**>, такие что их соединяет дуга с данной меткой. Этот метод индексирования был применен вкупе с языком запросов Xpath [1] и значительно превосходил по скорости выполнения запросов метод, внедренный в системе Lore. Его минус — достаточно большой размер самого индекса.

Одно из последних предложений по индексированию XML было сделано в работе [3]. Каждая метка дуги кодируется одним символом некоторого бесконечного алфавита, после чего пути из корня в дереве записываются как строки из этих символов, разделенных точками. Эти строки индекируются как ключи с помощью *Index Fabric* (модификация Patricia Trie), которая использует сжатие ключей и хорошо подходит для хранения на диске большого количества длинных и сложных строк.

В ходе выполнения запроса пути в нем преобразовываются в строки с помощью того же кодирования, после чего осуществляется поиск в *Index Fabric*. Авторы отмечают, что при использовании символов-множителей в запросе они осуществляют полный обход поддерева, что значительно увеличивает время выполнения запроса. В то же время можно использовать *refined paths* для часто повторяющихся запросов, то есть некоторые часто используемые пути кодировать всего одним символом.

3 Индексная структура

В этой работе за основу индексной структуры взята *Index Fabric*. Но в отличие от нее, метки дуг не кодируются символом бесконечного алфавита, а рассматриваются как слова над некоторым конечным алфавитом (например, ASCII или Unicode). В качестве базового индексного механизма взято сжимающее ключи B-дерево вместо Patricia Trie.

3.1 Модель

Здесь и далее слабоструктурированные данные ограничены рамками графовой модели XML, метки дуг в графе являются словами над некоторым конечным алфавитом Σ (например, ASCII или Unicode). В качестве разделителя меток дуг в пути используется символ " ." (точка), при этом предполагается, что " ." $\notin \Sigma$.

Пути обозначаются как

$$l_1.l_2 \ldots l_s$$

где $l_1, l_2, \ldots, l_s$ — метки дуг.

Решаемая задача: по данному XML-документу, представленному как множество путей, и заданному регулярному выражению пути найти все пути в документе, удовлетворяющие этому регулярному выражению.

3.2 Построение индекса

На первом шаге все множество путей представляется как множество строк с некоторым лексикографическим порядком. Эти пути являются ключами, которые хранятся

в сжимающем B-дереве (разумным представляется использование именно сжимающего B-дерева, чтобы уменьшить размер хранимого индекса). Данная индексная структура называется *Pindex*.

Затем каждой метке дуги сопоставляется число (*Depth*), вычисляемое как максимальная глубина этой дуги в дереве, считая от корня. В *Pindex* каждому узлу приписывается его глубина в OEM-дереве (*NDepth*), вычисляемая как глубина текущей метки дуги в данном пути. Мы полагаем, что все длины и глубины отсчитываются с нуля, т.е. путь, состоящий из одной дуги, имеет длину ноль.

И, наконец, строится дополнительный индекс (*Cindex*), с помощью которого по узлу в индексном B-дереве мы можем получить все листья, которые являются его потомками.

3.3 Использование индекса для выполнения запросов

При описании запросов мы пользовались таким синтаксисом для регулярных выражений:

1. Точка после бэкслэша (“ \.”) обозначает любой символ из Σ.
2. Знак вопроса (“ ?”) обозначает ни одного или один экземпляр стоящего перед ним символа.
3. Плюс (“ +”) обозначает один или более экземпляров стоящего перед ним символа.
4. Звездочка (“ *”) обозначает любое количество экземпляров стоящего перед ним символа.

Все запросы, описываемые регулярными выражениями, мы разбиваем на два класса: содержащие символы-множители (“ +” или “ *”) и не содержащие их. Соответственно и подход при поиске путей будет различным для этих двух классов.

3.3.1 Запросы без символов-множителей

При поступлении запроса без символов-множителей путь в запросе преобразуется в ключ-строку, поиск которой и осуществляется в *Pindex*. После нахождения листьев, которые удовлетворяют этому ключу, делается проверка пути, которому соответствует лист, на предмет соответствия выражению в запросе (ее необходимо осуществлять, так как мы используем сжимающее B-дерево). Если в запросе встретился символ “ ?”, то запрос разбивается на два соответствующих подзапроса (с необязательной частью и без нее).

3.3.2 Запросы с символами-множителями

Вполне очевидно, что запрос вида **“ a+b”** можно переписать как запрос **“ aa*b”**, поэтому в дальнейшем в качестве символа расширения будет рассматриваться только символ “ *”. Более того, чтобы упростить восприятие, в этой работе будут обсуждаться запросы с единственным символом-множителем.

Все запросы с одним символом-множителем можно разбить на три подкатегории:

1. Запросы вида **“ a (t)*”** (здесь и в дальнейшем предполагается, что прописными буквами латинского алфавита обозначается часть, не содержащая символов-множителей).

2. Запросы вида “ **a (t)* b** ”.

3. И запросы вида “ **(t)* b** ”.

Заметим, что интерес представляют в основном такие запросы, где **t=(\.)**, поэтому мы сконцентрировали наши усилия на изучении именно их. Часть запроса, которая стоит до символа-множителя, называется префиксом запроса, после — суффиксом запроса.

Запросы первого рода обрабатываются просто: осуществляется поиск в *Pindex* по началу ключа “ **a** ”, затем берется тот узел, в котором поиск остановился, осуществляется сравнение начала пути в запросе и начала пути, которому соответствует этот узел. При получении положительного ответа при сравнении используется *Cindex* для получения всех путей, которые соответствуют данному запросу.

Запросы второго рода начинают обрабатываться как и запросы первого рода, с поиска начала ключа в *Pindex*. После того, как поиск останавливается, получается, что мы должны обработать запрос третьего рода, только не с начала дерева, а в каком-то поддереве. Поэтому сразу перейдем к рассмотрению запросов третьего рода.

Поиск по запросу третьего рода начинается с того, что в части “ **b** ” запроса для каждой метки дуги берется *Depth* (если какая-то из меток не существует в дереве, то ответ на запрос сразу отрицательный). Затем для каждой из дуг в суффиксе запроса вычисляется величина, называемая относительной глубиной (*RDepth*) метки дуги в суффиксе. Вычисляется она как глубина метки дуги относительно начала суффикса запроса, причем первая метка в суффиксе получает значение $RDepth = 0$. После этого вычисляется глубина запроса, обозначаемая *QDepth*:

$$QDepth = \min_{l \in (Suffix)} (Depth(l) - RDepth(l)) + Length(Suffix)$$

Попутно проверяется, что

$$\forall l \in Suffix : (Depth(l) - RDepth(l) \geq 0)$$

(иначе ответ на запрос тоже отрицательный).

Затем строится недетерминированный конечный автомат, который принимает строки, соответствующие регулярному выражению в запросе. Этот автомат содержит стек из состояний для каждого узла, поэтому когда автомат отвергает какую-то строку, осуществляется откат по стеку в соответствующее состояние и не проверяется заново общая часть строки. По сути этот автомат аналогичен реализации абстрактного логического вычислителя в языке Пролог: с поиском решения по дереву поиска в глубину и откатом на следующую альтернативу при неудаче.

После этого начинается обход дерева (поддерева) в глубину. При попадании в узел B-дерева его *NDepth* сравнивается с *QDepth*. Если $QDepth < NDepth$, то глубже можно не спускаться, так как эта ветка точно не содержит искомого пути (так как максимальная длина искомого пути равна *QDepth*). При достижении листа путь, ему соответствующий, пропускается через автомат на предмет соответствия регулярному выражению из запроса.

4 Заключение и будущая работа

В данной работе представлен метод индексирования XML документов, поддерживающий использование регулярных выражений в запросах. Подобные структуры

необходимы для эффективной реализации таких средств спецификации запросов, как Xpath и Xquery, а также алгебры XML, разрабатываемых W3C.

Предложенная струтктура, очевидно, имеет существенные преимущества по сравнению со структурами, не поддерживающими выражений над путями, однако может уступать им в тех случаях, когда пути специфицированы полностью или производится поиск только по значениям без использования выражений над путями.

Наиболее эффективно предложенная структура работает в том случае, когда количество различных меток велико, и, вероятно, работает менее эффективно для документов с более регулярной структурой.

Детальный анализ эффективности предложенного метода, сопоставление с другими методами индексирования, а также результаты экспериментов на больших объемах данных, будут включены в полный текст статьи.

Список литературы

[1] W3C recomendation. XML path language (Xpath) 1.0. `http://www.w3.org/TR/xpath`, 1999.

[2] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997. Also `http://www-db.stanford.edu/pub/papers/lorel96.ps` from Stanford DB group on-line publications `http://www-db.stanford.edu/pub/`.

[3] F. Cooper, Neal Sample, Michael J. Franklin, Gisli Hjaltason, and Moshe Shadmon. Fast index for semistructured data. In *Proc. VLDB 2001*, pages 341–350, 2001.

[4] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Twenty-Third International Conference on Very Large Data Bases*, pages 436–445, 1997.

[5] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(3):54–66, September 1997.

[6] J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajamaran. Indexing semistructured data, 1998.

[7] Tova Milo and Dan Suciu. Index structures for path expressions. In *ICDT*, pages 277–295, 1999.

[8] Flavio Rizzolo and Alberto Mendelzon. Indexing xml data with toxin.

[9] Extensible Markup Language (XML) 1.0 (Second Edition). `http://www.w3.org/TR/2000/REC-xml-20001006`.