УДК 004.75

МОДЕЛИ И МЕТОДЫ ПОИСКОВОЙ ОПТИМИЗАЦИИ: СТРУКТУРНАЯ ПРЕДОПТИМИЗАЦИЯ ДАННЫХ

Распопин Н.А.
Научный руководитель — д.т.н. Ковалев И.В.
Сибирский федеральный университет

Сегодня редко можно встретить человека, который не использовал бы в своей работе или повседневной жизни поисковые технологии. Поиск информации в информационных сетях, каким мы его себе представляем сегодня, начал зарождаться задолго до появления глобальной сети. Прообразом первых сетевых поисковиков можно считать картотечную систему на подобие той, что мы встречали в библиотеках, когда большое количество документов классифицируются по особому признаку: это может быть жанр произведения, отрасль науки, автор, название произведения или упорядочивание по алфавиту. Классификация может быть одноуровневая, когда мы производим ее только по одному признаку или многоуровневая, когда признаков несколько, например, мы классифицируем всю литературу по отрасли науки, а потом каждую отрасль науки классифицируем по направлению или по автору и так далее. Система каталогов существовала в бумажном виде долгое время, а в последствии она была перенесена в электронный вариант, в Интернете, в частности, можно было встретить сайты-каталоги, где все многообразие web страниц было строго сортировано по тематике, и для того, чтобы найти «нужный» сайт, приходилось просматривать десятки, а то и сотни ресурсов. С тех пор в сфере поисковых технологий изменилось очень многое

Первые поисковые системы сталкивались с большим количеством проблем, прежде всего технического характера. В то время, как математические модели позволяли реализовывать достаточно сложные по содержанию алгоритмы, которые могли обеспечить приемлемый для того времени уровень сервиса, аппаратные средства не могли предоставить платформы для реализации подобного рода идей. В значительной степени не хватало процессорной мощности для обеспечения производительности, однако основной проблемой была нехватка памяти как оперативной для реализации самого процесса поиска, так и дисковой для хранения пространства данных. Разработчики были вынуждены прибегать к многократной оптимизации, что сказывалось на качестве конечного продукта – приходилось чем-то жертвовать. С развитием вычислительной техники подобного рода эти проблемы ушли на второй план, уступив место новым. Главная из которых – огромный объем информации, который, ко всему прочему, имеет очень низкий уровень формализации, что в значительной степени затрудняет поиск в пространстве данных. Под уровнем формализации понимается степень смысловой и любой иной неоднозначности данных, например, визуальной. Можно сказать, что пространство чисел имеет гораздо более высокий уровень формализации, нежели пространство текстовой информации. Число 15 в одном документе и число 15 в другом документе несут на себе одну и ту же смысловую и визуальную нагрузку вне контекста, в то время, как слово «топорище» вне контекста содержит неоднозначность – это может быть «большой топор» или «ручка топора». Большое пространство данных содержит в себе еще одну неоднозначность - лингвистическую - это грамматические, синтаксические, семантические ошибки, которые искажают пространство данных, приводя, в конечном итоге, к ошибочным результатам работы поисковой системы.

Лавинообразное накопление информации требует совершенно новых подходов к решению проблемы поиска по информационному пространству. Под информационным пространством понимается вся совокупность данных в цифровом виде, доступная посредством различных носителей информации, а также местных локальных и глобаль-

ных сетей. Для эффективной работы алгоритмов обработки и поиска информации, прежде всего, необходимо разработать структуру представления данных, которая отвечала бы нескольким основным требованиям: максимальное быстродействие поисковых алгоритмов, простота реализации по структуре данных алгоритмов синтаксического, морфологического и семантического анализов. Разработка структуры представления данных является достаточно сложной и ответственной задачей, от успешной реализации которой зависит дальнейшее функционирование всей системы. Структура представления данных представляет собой тот фундамент, на котором по кирпичику укладываются различные модули и алгоритмы системы, обеспечивающие выполнение запросов конечных пользователей, т. е. нас с вами.

Акцентируем свое внимание на первом требовании, озвученном выше – максимальное быстродействие поисковых алгоритмов. Самым оптимальным вариантом является случай, при котором на поиск подстроки в строке требуется не больше операций, чем содержит подстрока символов. Приведем пример: нам требуется найти слово «объем» в одном или нескольких документах. Само слово содержит 5 символов, документ при этом может содержать неограниченное количество символов. Рассмотрим представленную на рис. 1 структуру данных, которая образована из исходного документа путем проведения операции предварительной поисковой оптимизации. Данная структура представляет собой дерево анализа, состоящее из несовпадающих слов исходного документа, связанных между собой *определяющими* узлами ветвей и основным корнем дерева. Основной корень дерева на рисунке отмечен фиолетовым цветом. Он представляет собой указатель на первые буквы слов, которые содержатся в исходном документе. *Определяющие* узлы ветвей отмечены зеленым цветом, они представляют собой узлы в символьной цепочке дерева (их наличие говорит о присутствии однокоренных слов или различных словоформ одного слова), у которых имеется хотя бы один потомок и которые остаются в дереве, если все дочерние узлы удалить. Если же при удалении дочерних узлов *определяющий* узел тоже удаляется, то такой узел будет называться *транзитным*. Желтым цветом отмечены узлы или листья дерева, которые не имеют потомков – *конечные* узлы.

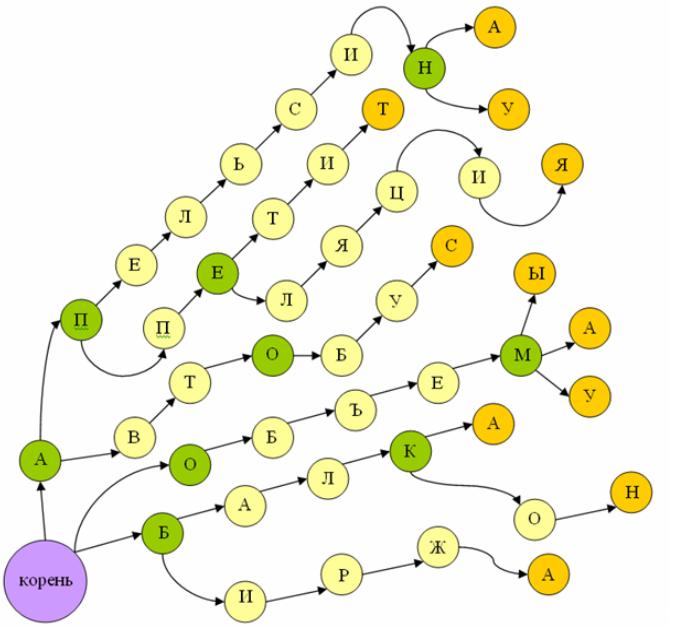

Рис. 1. Структура данных поисковой системы

Рассмотрим более подробно правило построения представленной на рис. 1 структуры. Исходный документ разбивается на отдельные составляющие – слова (элементы), затем каждое слово побуквенно накладывается на уже имеющуюся структуру так, как показано на рис. 1. Например, если в тексте встречается 10 раз слово «объем», 15 раз слово «объему», 20 раз – «объемы» и 25 раз – «объема», которые все вместе содержат в себе 410 символов - в структуре они займут всего лишь 8 символов.

На рис.2 представлена процедура поиска слова «объема» по структуре данных,

она заключается в последовательном переходе от узла-корня структуры к узлу-потомку с посимвольным сравнением содержания соответствующего узла с буквой в слове, отстоящей по порядку от начала слова на величину уровня узла потомка. Введем функцию $z(t)$, которая в каждый момент времени t будет обозначать букву в искомом слове, находящуюся на $t+1$ месте от начала. Введем функцию $p(z(t+1),r(t))$, которая будет возвращать указатель на потомка узла структуры, если у текущего узла есть потомок, который содержит $z(t)$, и будет возвращать 0, если у узла нет потомков или ни один потомок не содержит $z(t)$, где $r(t)$ – указатель на текущий узел в момент времени t.

В момент $t=0, z(1)="o", p(z(1),root)="o"$ функция вернула указатель на узел, который является дочерним для корня и содержит искомую букву. В момент $t=1, z(2)="б", p(z(2),"o")="б"$. В момент времени $t=5, z(6)="а", p(z(6),"м")="а"$. В момент времени $t=6, z(7)=NULL$ – это означает, что проверка закончилась, и все буквы слова присутствуют в структуре данных в том порядке, в котором они идут в слове. Для того, чтобы подтвердить нахождение слова в документе, достаточно, чтобы последний узел был *конечным* или *определяющим*. Если операция поиска закончится на *транзитном* узле, то это недостаточное условие, и значит искомого слова в документе нет. В нашем случае это *конечный* узел – искомое слово найдено.

Итак, для полученной на рис. 1 структуры данных мы определили поисковый алгоритм, который позволяет найти в ней искомое слово. В случае поиска подстроки или более крупного фрагмента текста, необходимо проверять его наличие поэлементно (рассмотрение этого момента не является целью данной статьи). В данном случае можно показать, что максимальное количество операций $P_{max}(x)$ для поиска слова занимает не больше операций, чем длина $N(x)$ самого слова плюс единица:

$$P_{max}(x) = N(x)+1$$

$$P(x) <= N(x)+1 \quad ,$$

$$P(x) > 0$$

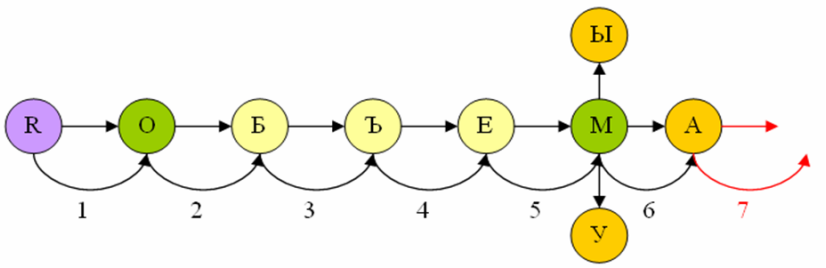

Рисунок 2 Процедура поиска по структуре данных

Выражение $0 < P(x) <= N(x)$ говорит о том, что количество операций, затрачиваемое на поиск слова, может быть меньше, чем $P_{max}(x)$, однако всегда больше 0. Это может быть в том случае, если искомого слова в документе нет. Выражение $P(x) > 0$ следует из того, что для получения отрицательного результата от поискового алгоритма необходимо произвести, как минимум, одну операцию – определение нулевого указателя на узел с первой буквой слова. Математически поисковый алгоритм будет выглядеть следующим образом:

$$\sum_{i=0}^{(N(x)-1)} p(z(i+1),r(i)) = \sum_{i=0}^{(N(x)-1)} z(i+1).$$

Если равенство соблюдается, то элемент присутствует в структуре, если нет – элемент в структуре не содержится.

Мы рассмотрели простейший случай поискового запроса. Фактически это бинарная операция по определению присутствия или отсутствия слова в тексте. На прак-

тике потребности в поисковых алгоритмах довольно редко ограничиваются столь примитивной формой. Практически всегда требуется определить наличие в тексте группы слов, выражений или даже целых текстовых фрагментов, также может потребоваться наложить некоторые ограничения на искомые элементы: это может быть дата их создания, дата последнего редактирования, расположения в документе, регистра символов и так далее. В конечном счете, может получиться поисковый запрос, содержащий в себе десятки и сотни элементов и ограничений, однако, сколь сложным бы он ни был, его решение всегда сводится к разбиению на простейшие составляющие элементы, каждый из которых обрабатывается независимо друг от друга. Полученные данные от каждого элементарного поискового запроса результируются, согласно внутренней логике системы, в единое целое, которое представляет собой ответ системы на первоначальный поисковый запрос. С учетом вышесказанного наш алгоритм будет выглядеть следующим образом:

$$\sum_{k=1}^{N(y)} \sum_{i=0}^{(N(x_k)-1)} p(z(i+1), r(i)) = \sum_{k=1}^{N(y)} \sum_{i=0}^{(N(x_k)-1)} z(i+1),$$

где $N(y)$ - количество элементарных запросов, на которые разбит первоначальный x.

Несмотря на кажущуюся простоту и легкость работы со структурой на рис 1., в ней есть существенный недостаток. В подавляющем большинстве случаев пользователя интересует не столько положительная или отрицательная реакция поисковой системы, сколько получение конкретного документа или указания места в документе, которому соответствует его поисковый запрос. Предложенная на рис. 1 структура в общем виде не удовлетворяет условиям, при которых можно было бы реализовать дополнительный функционал. Более того, преобразование документа в структуру необратимо, т. е. произвести обратное преобразование и получить документ из структуры невозможно – это означает, что не существует однозначного отображения оптимизированной структуры в исходный документ. Соответственно, получив по ходу работы поискового алгоритма элемент в оптимизированной структуре, анализировать его в связи с первоначальным документом будет нельзя. Для устранения указанных ограничений нам понадобится ввести некоторые изменения в существующий алгоритм оптимизации, а именно - дополнительную структуру данных, которую мы назовем *расширенной базой.* Структура расширенной базы должна разрабатываться исходя из потребностей поисковой системы. Чем больше возможностей будет иметь поисковая система, тем сложнее и больше будет расширенная база.

Положим, наша система должна будет иметь возможность перевести курсор на позицию, в которой находится искомый элемент, если таковой существует. Также должна быть реализована возможность поиска подстроки, состоящей из произвольного количества элементов. Под элементом понимается отдельное слово или любое другое цифробуквенное выражение, отстоящее от других на один или более знак разделитель (по умолчанию разделителем будем считать пробел). Для того, чтобы иметь возможность перевести курсор на позицию элемента необходимо, чтобы в структуре была представлена информация о местоположении каждого отдельного слова. Для всех узлов структуры, которые являются *конечными* или *определяющими,* создается отдельная таблица, в которой хранится информация о каждом элементе.

В нашем случае необходимо хранить следующую информацию об элементе:

- принадлежность элемента к документу $\varepsilon \in \mathrm{P}$, где ε - индекс элемента (он же индекс таблицы), P - индекс документа. Следует отметить, что $\varepsilon = \mu$, где μ - индекс узла. Индекс узла назначается *конечным* и *определяющим* узлам структуры, транзитным узлам индекс не назначается;
- индекс предыдущего и индекс следующего элемента в данном документе $\varepsilon_{i-1} < \varepsilon_i < \varepsilon_{i+1}$. Эти данные нам понадобятся для выполнения сложных запросов

и осуществление лингвистического анализа текста;

- существенное положение относительно начала документа элемента ε в документе P обозначим как θ. По этому параметру, исходя из $\varepsilon \in \mathrm{P}$, можно будет однозначно восстановить исходный документ из оптимизированной структуры. Таким образом, ограничение необратимости преобразования, о котором мы говорили выше, снимается.

Содержание расширенной базы напрямую зависит от требуемой функциональности системы. В случае дополнительных требований информативность расширенной базы может быть увеличена путем добавления в нее необходимых полей в таблицу элемента, это может быть время создания документа, лингвистические характеристики: род, число, падеж и так далее. Чем информативнее будет расширенная база, тем больше будет возможности для проведения анализа, тем качественнее может быть работа всей поисковой системы в целом.