УДК 004.822, 004.021
DOI: 10.17586/0021-3454- 2017-60-10-932-939

МЕТОД БЫСТРОГО ПОИСКА УЗЛОВ СЕМАНТИЧЕСКОЙ СЕТИ ПО ТОЧНОМУ СОВПАДЕНИЮ СЛОВОФОРМЫ

А. В. Покид[1], С. В. Клименков[2], Е. А. Цопа[2],
С. А. Жмылёв[3], Н. М. Ткешелашвили[4]

[1]*ООО „СВД Встраиваемые Системы“, 196128, Санкт-Петербург, Россия*
[2]*Университет ИТМО, 197101, Санкт-Петербург, Россия*
[3]*ООО „Сервиком“, 195197, Санкт-Петербург, Россия*
[4]*ООО „Элком“, 195197, Санкт-Петербург, Россия*

Разработка и использование онтологий является необходимым элементом анализа текстов на естественном языке. При потоковой обработке текстов время поиска в онтологии является критичным параметром для обработки большого объема данных. Предложен метод поиска словоформ по точному совпадению, согласно которому сначала словоформы слов из онтологии разбиваются на части определенной длины (*x*-граммы), по разработанному алгоритму вычисляется индекс *x*-граммы и узлы словоформ организуются в префиксное дерево, каждый уровень которого представлен в виде массива. Индекс *x*-граммы используется в качестве ключа. Для обеспечения компактности хранения выполняется операция сжатия набора разреженных массивов. Алгоритм словарного поиска, в свою очередь, разбивает искомый токен (слово) на соответствующие ему *x*-граммы, вычисляет индекс каждой части и по полученным индексам в массивах, соответствующих каждому из уровней префиксного дерева, находит словоформу в онтологии последовательной выборкой. Разработанное программное обеспечение показывает для тестового набора русских словоформ скорость поиска выше на 36—50 %, по сравнению с Google dense hashmap, а объем занимаемой памяти на 12 % меньше, чем в Google sparse hashmap. Разработанный метод применим для словарного поиска по редко изменяемым наборам искомых словоформ, таким как онтология, построенная на базе Викисловаря.

Ключевые слова: *поиск в словаре, онтология на основе Викисловаря, префиксное дерево, слогоделение, n-граммы, хэш-таблицы, точный поиск*

Введение. Лавинообразное увеличение количества информации, представленной в электронном виде, требует разработки систем смыслового или семантического анализа текста на естественном языке. Неотъемлемой частью таких систем являются онтологии, обычно представляющие собой сеть связанных понятий предметной области. Одной из задач авторов является разработка и использование в семантическом анализе онтологии [1, 2] на базе русскоязычного Викисловаря (http://ru.wiktionary.org/wiki). Следует отметить, что, несмотря на рост возможностей средств вычислительной техники, увеличение производительности процессоров и объемов хранимых данных, скорость обработки текстовой информации в задачах смыслового анализа все еще остается неудовлетворительной.

Обработка текстов на естественном языке выполняется в несколько этапов. Сначала происходит чтение текста и разбивка его на токены. Токен — это минимальная линейная единица членения предложения [3]. Следующим этапом является сопоставление полученных токенов с узлами типа „словоформа“ разрабатываемой семантической сети. Если токен представлен словоформой, не содержащей орфографических ошибок (большинство современных текстов пишутся с использованием специализированных корректоров правописания и содержат небольшой процент ошибок), это сделать достаточно просто, однако анализ большого количества таких операций требует существенного времени. Для словоформ с орфографическими

ошибками при потоковом анализе необходимо подобрать наиболее подходящую словоформу из семантической сети, однако это требует предварительного выявления факта наличия ошибки в токене.

Целью настоящей работы является разработка метода быстрого сопоставления токенов без ошибок с узлами семантической сети, позволяющего обнаруживать ошибки на ранних стадиях.

Алгоритмы словарного поиска. Поиск узла семантической сети представляет собой задачу словарного поиска, которая может быть решена при помощи ассоциативных массивов. Узким местом алгоритмов построения ассоциативных массивов является увеличение времени поиска по мере заполнения словаря большим количеством элементов [4]. На практике наиболее часто используются деревья поиска и хэш-таблицы [5]. Для сравнения скорости поиска в настоящей статье используется реализация красно-черных деревьев, лежащая в основе таких структур данных, как map и set библиотеки STL.

Хэш-таблицы строятся на основе некоторой хэш-функции $h(\text{key})$, отображающей множество всех ключей в множество доступных ячеек памяти для хранения значений. Вследствие того что объем исходного множества возможных ключей, как правило, превышает объем доступной памяти, а множество используемых ключей, как правило, неизвестно, то возникновение коллизий (когда у разных ключей значение хэш-функции совпадает) неизбежно. Существуют различные стратегии разрешения коллизий, например, использование хэш-таблиц с открытой адресацией [6] и хэш-таблиц с разрешением коллизий с помощью цепочек [7]. Как было показано в [8], использование хэш-таблиц может быть эффективно для хранения словаря английских слов.

Одной из наиболее эффективных хэш-таблиц является библиотека Google sparehash package, время поиска с ее помощью является одним из минимальных [9, 10]. Эта библиотека выбрана в качестве эталонной в настоящем исследовании. Алгоритм Dense hashmap показывает лучшие времена поиска, а sparse hashmap — лучшие характеристики компактного представления служебных структур в памяти.

Другим эффективным способом построения словарного индекса является применение префиксных деревьев [10]. Время поиска в них зависит не от размера словаря, а от максимальной длины ключа. В ходе проведенных экспериментов [11] было выявлено, что для поиска слов длиной 16 символов в словаре использование префиксных деревьев дает лучший, по сравнению с бинарными деревьями, результат.

Высокую скорость обеспечивают и аппаратные средства, как например в [12], однако они требуют дополнительных специализированных процессоров, что приводит к увеличению стоимости поиска.

Разработанный метод словарного поиска. Обновление производного семантического графа в текущей реализации из Викисловаря происходит раз в сутки, поэтому алгоритмической сложностью операций вставки и обновления, которые также используются для классических структур данных словарей [15], можно пренебречь.

Основываясь на допущении о том, что доля слов с опечатками или орфографическими ошибками в тексте невелика, будем полагать, что алгоритм поиска словоформы в семантической сети должен быстро найти точное совпадение лексемы исходного текста с существующей словоформой в сети.

Если во время поиска будет обнаружено, что лексема отсутствует (например, написана с ошибкой), необходимо произвести поиск с учетом ошибок и найти наиболее подходящий узел словоформы. Эта задача также может быть решена с применением алгоритмов и структур данных, предложенных в настоящей статье.

Организация токенов в префиксное дерево. Наиболее эффективный способ организации поиска заключается в прямом отображении числового представления строки в узел семантической сети. Однако это требует $(k \cdot N^n)$ байтов памяти (где k — размер хранимой

структуры данных, N — мощность алфавита, n — максимальная длина слова). Для уменьшения требуемого объема можно разбить поисковый токен, представляющий словоформу, на несколько частей, каждая из которых может быть использована в качестве индекса в массиве, а последовательность частей объединить в префиксное дерево согласно выбранному способу деления токена на части. Это позволит произвести поиск максимум за $|f(t)|$ обращений к массивам (f — функция разбиения токена, а t — токен, являющийся ключом для поиска в словаре). Такой подход позволяет уменьшить занимаемый объем памяти.

Организация декомпозиции токена. Выбор способа декомпозиции токена напрямую влияет как на размер структур данных, используемых для поиска, так и на время поиска в словаре. При использовании префиксных деревьев максимальное время поиска по ключу будет равно максимальной высоте дерева.

В качестве подходов были выбраны:

— разбиение на 3-граммы (ровно три символа токена);

— разбиение на 4-граммы;

— разбиение фонетическим слогоделением [16];

— разбиение по морфологическим правилам переноса.

Для разбиения по правилам переноса были адаптированы алгоритм и база данных [17] программного продукта LaTeX. В качестве набора словоформ были использованы электронные версии грамматического словаря [18] и открытого корпуса русского языка [19].

Минимальную высоту дерева (h=7) ожидаемо показало деление на 4-граммы. Декомпозиция слогоделением (h=11) и морфологическим переносом (h=12) дает большую высоту дерева, поскольку такие слоги могут включать переменное (от 1 до 8) количество символов. Однако эти разбиения в дальнейшем можно использовать для определения токенов с ошибками и завершать точный поиск, а также для исправления некоторого класса типичных ошибок слов.

Части токена, которые получаются при декомпозиции, когда способ деления не уточнен или не имеет значения, будем называть *x-граммами*.

Формирование индекса *x*-грамм. Вследствие трудностей, возникающих при манипуляции строковыми типами данных, одним из базовых понятий предлагаемого алгоритма является функция, осуществляющая отображение строки в пространство натуральных чисел:

$$I(s) = \sum_{i=0}^{|s|} \left((s_i - A_0) |A|^i \right),$$

где I — числовое представление токена; s — строка, для которой вычисляется численное представление, в частности x-грамма; A — алфавит языка; $|s|$ — длина входной строки; $|A|$ — число символов в алфавите; A_0 — первый символ алфавита.

Поскольку операция преобразования строки в индекс — одна из наиболее частых при работе со словарем, она должна выполняться за минимальное время.

Размещение x-грамм в массиве свойства естественного языка приводит к локализации групп подмножеств индексов, что создает высокую разреженность массива. Как следствие, хранение множества индексов требует существенной оперативной памяти, что делает размещение всех уникальных индексов в едином разреженном массиве невозможным.

В результате частотного анализа деления на слоги выявлено, что около 87 % всех слогов словаря меньше 5 символов. Также было обнаружено, что пространство, образуемое оставшимися частями x-грамм, невелико, это позволяет хранить „хвостовые части“ x-грамм (с пятого по восьмой символ) в одной таблице. Такой способ разбиения позволит находить большую часть x-грамм за одно обращение к памяти, а более длинные x-граммы — за два.

Получение уникального номера *x*-граммы. Для хранения уникальных числовых идентификаторов (номеров) *x*-грамм используются две таблицы (рис. 1, *а*). Первая состоит из дескрипторов, содержащих номер для *x*-грамм размером менее пяти символов (например, номер *x*-граммы ‘ту’ $P_{ту}$=18 733), ссылку на таблицу второго уровня, а также значение минимального и максимального элемента дочерней таблицы. Таблица второго уровня содержит исключительно уникальные номера. Минимальный и максимальный элементы дескриптора задают нижнюю и верхнюю границы номеров в дочерних таблицах. Исходя из особенностей описанных выше структур данных возможны два вида операций поиска номера *x*-граммы:

1) для *x*-грамм, размер которых составляет менее пяти символов — с помощью формулы преобразования строки в ее числовое представление вычисляется индекс (I(‘ту’)=681), и по данному индексу из таблицы извлекается элемент, содержащий номер;

2) *x*-грамма размером более четырех символов разбивается на две подстроки таким образом, что в первой подстроке содержатся первые четыре символа, а во второй — оставшиеся. Первая подстрока преобразуется в индекс для извлечения дескриптора из таблицы первого уровня. Дескриптор хранит ссылку на таблицу второго уровня, содержащую искомое значение. В качестве индекса для поиска в таблице второго уровня используется численное представление второй подстроки *x*-граммы.

Кроме того, описанная схема позволяет сократить время получения ответа благодаря завершению поиска еще на раннем этапе в случае, если ключ в словаре не найден, опуская этап анализа дерева номеров *x*-грамм.

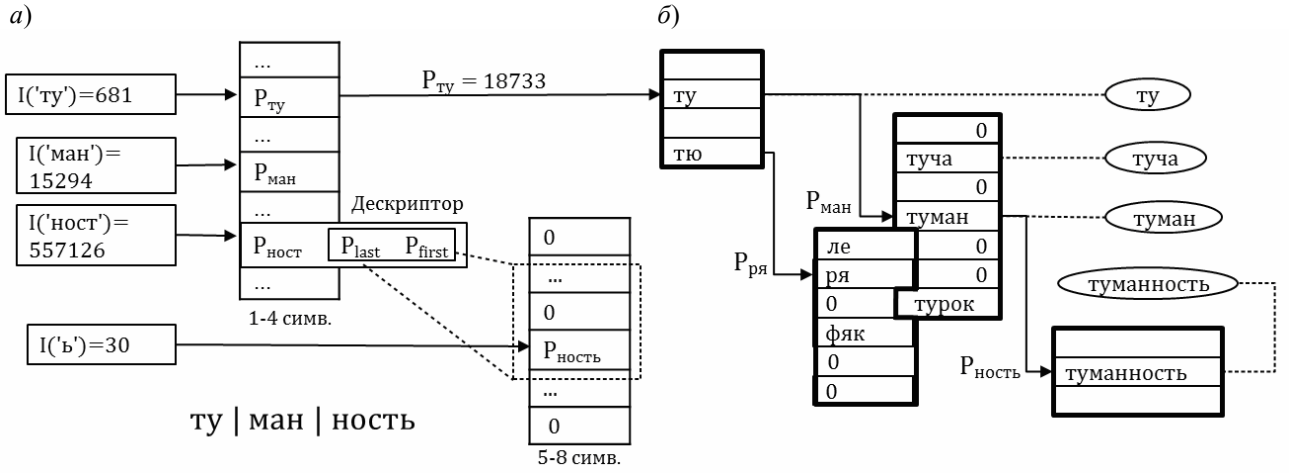

Рис. 1

Построение префиксного дерева и поиск в нем. Полученная после преобразования исходного токена последовательность номеров *x*-грамм сохраняется в префиксное дерево (рис. 1, *б*). Узлы префиксного дерева хранятся в структуре данных, схожей с рассмотренной выше. Каждый узел аналогичным образом представлен дескриптором, а дочерние узлы содержатся в разреженном массиве. Одной из особенностей префиксного дерева для хранения слов является наличие в дескрипторе дополнительной ссылки на хранимые данные. Таким образом, алгоритм получения данных по ключу включает следующие шаги.

1. Преобразуем исходный ключ в массив индексов.

2. Положим $k = 1$.

3. Извлечем k-й дескриптор путем сложения адреса следующей таблицы с индексом k и вычитания минимального элемента, хранящегося в дескрипторе.

4. Повторим шаг 4, пока k не равно числу индексов.

5. Извлечем данные из текущего дескриптора и выйдем.

Полученные структуры хранения словаря используют большой объем памяти и сильно разрежены. В отличие, от например дедупликации [19], операция вставки и удаления элементов дерева не производится. Следовательно, можно провести сжатие полученных структур,

при котором для уменьшения объема занимаемой оперативной памяти все таблицы группируются по их соответствию позициям в множестве индексов. Затем внутри созданных групп с помощью изменения смещений таблиц в соответствующих им дескрипторах происходит компоновка и вставка таблиц друг в друга (рис. 1, *б*, слова тюря, тюфяк, турок). Полученный результат записывается в общую таблицу.

Результаты. Алгоритм поиска и комплекс вспомогательных программ были реализованы на языке C++, выполнены замеры скорости выборки информации из разработанных структур для *x*-грамм, основанных на фонетическом слогоделении, а также 3- и 4-грамм. Для сравнения были выбраны стандартные реализации алгоритмов std::map и std::unordered_map, а также входящие в пакет Google sparehash package google::dense_hash_map и google::sparse_hash_map.

Аппаратная конфигурация системы, на которой производились эксперименты: процессор — Intel Pentium P6200@2.13ГГц, два ядра, размер кэш-памяти — 3072 КБ, размер кэш-строки — 64 байта. Для проверки скорости поиска создан общий словарь размером 3 004 310 словоформ, проведен количественный анализ словоформ от числа символов в них.

Оценка объема памяти, занимаемой структурами хранения, выполнялась на основании размера памяти, занимаемой процессом после выполнения циклов занесения данных в структуры и их последующего поиска. Реализации предлагаемого метода для *x*-грамм демонстрируют объем занимаемой памяти в среднем 629 МБ, что на 12 % меньше, чем у Google sparse hashmap (715 МБ). Объемы занимаемой памяти *x*-грамм практически не различаются между собой, потому что сжатие структур осуществляется одинаковым образом.

Замеры времени поиска проводились для подмножества словоформ словаря. В первой группе замеров (рис. 2, *а*, где *1* — 3-граммы, *2* — 4-граммы, *3* — слоги, *4* — std::unordered_map, *5* — google:dense_hashmap, *6* — google:sparse_hashmap, *7* — std::map) на вход алгоритмов подавались слова, содержащие заранее заданное число символов, а во второй группе (рис. 2, *б*) — слова, содержащие меньшее или равное заданному количество символов, указанных по оси абсцисс. В начале замера получалась временная метка, затем подавалось заданное число словоформ, после этого вычислялось время поиска, нормированное по числу словоформ. Каждый из замеров проводился восемь раз, и результаты этих замеров были усреднены.

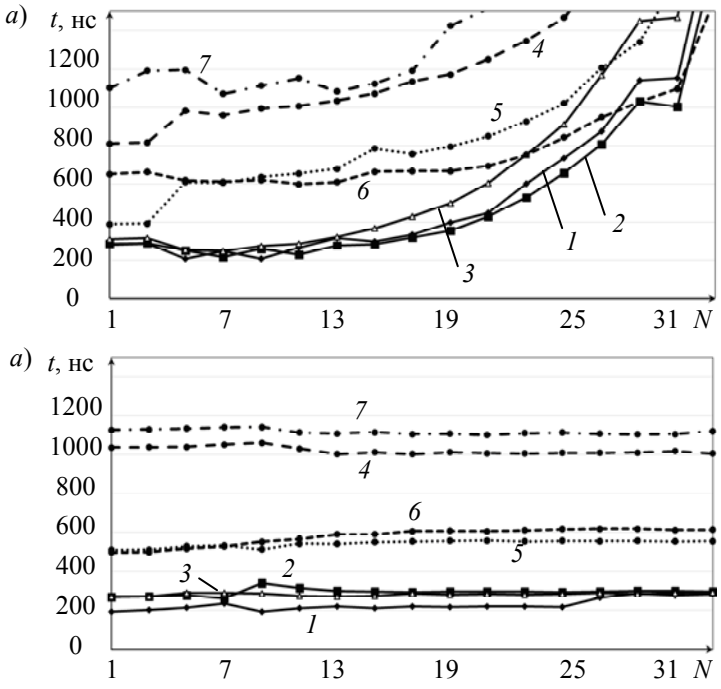

Рис. 2

Из рис. 2, *а* видно, что для всех исследованных алгоритмов время поиска резко возрастает для слов свыше 19 символов. При этом разработанный алгоритм показывают существенное преимущество по сравнению с аналогами — при типичном размере слова в 7—12 символов они быстрее dense hashmap более чем в 1,5 раза. Из рис. 2, *б* видно, как проявляют себя исследуемые алгоритмы с увеличением числа символов от 1 до *N*. Именно так происходит при анализе текстов на естественном языке. Время поиска одного токена в среднем составляет 200—290 нс в сравнении с 542 нс у dense_hash_map. Наилучшим из рассматриваемых оказался поиск по 3-граммам, его среднее время поиска для слов от 1 до 34 букв составило 227 нс.

Заключение. В статье предложен метод поиска узлов семантической сети по точному совпадению, которое имеет постоянную временную сложность наихудшего случая. Разработанный метод дал снижение времени поиска в 51 % по сравнению с выбранными для анализа. Следует заметить, что при небольшом количестве *x*-грамм в токене (можно интерпретировать как слоги в искомом слове) скорость поиска выше на 59 %.

Предварительно установлено, что использование простейших алгоритмов нечеткого поиска с использованием редакционных расстояний [20] между *x*-граммами с использованием разработанного префиксного дерева может привести к существенному уменьшению времени поиска с ошибками.

СПИСОК ЛИТЕРАТУРЫ

1. *Письмак А. Е., Харитонова А. Е., Цопа Е. А., Клименков С. В.* Метод автоматического формирования семантической сети из слабоструктурированных источников // Программные продукты и системы. 2016. № 3. С. 74—78.
2. *Klimenkov S., Tsopa E., Pismak A., Yarkeev A*. Reconstruction of Implied Semantic Relations in Russian Wiktionary // Proc. of the 8th Intern. Joint Conf. on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KDIR). 2016. Vol. 2. P. 74—80.
3. *Бочаров В. В., Грановский Д. В., Суриков А. В.* Вероятностная модель токенизации в проекте „Открытый корпус“ // Матер. 15-го науч.-практ. семинара „Новые информационные технологии в автоматизированных системах“. М.: Моск. гос. ин-т электроники и математики, 2012.
4. *Neyer M. P.* A Comparison of Dictionary Implementations. 2009 [Электронный ресурс]: <https://www.cs.unc.edu/~plaisted/comp550/Neyer%20paper.pdf>.
5. *Mehlhorn K. and Sanders P.* Algorithms and Data Structures. The Basic Toolbox. Springer, 2007. 285 p.
6. *Peterson W. W.* Addressing for Random-Access Storage // IBM J. of Research and Development. 1957. Vol. 1. P. 130—146.
7. *Dumey A. I.* Indexing for rapid random-access memory // Computers and Automation. 1956. Vol. 12, N 5. P. 6—9.
8. *Comer D. and Shen V. Y.* Hash-Bucket Search: A Fast Technique for Searching an English Spelling Dictionary Software // Practice and Experience. 1982. Vol. 12, N 7. P. 669—682.
9. *Silverstein C.* Google sparsehash package. 2010.
10. *Silverstein C.* Google sparsehash package performance [Электронный ресурс]: <http://goog-sparsehash.sourceforge.net/doc/performance.html 2010>.
11. *Acharya Anurag, Huican Zhu, and Kai Shen.* Adaptive algorithms for cache-efficient trie search // Workshop on Algorithm Engineering and Experimentation. Lecture Notes in Computer Science. Berlin—Heidelberg: Springer, 1999. Vol. 1619. P. 300—315.
12. *Thenmozhi M., Srimathi H.* An Analysis on the Performance of Tree and Trie based Dictionary Implementations with Different Data Usage Models // Indian J. of Science and Technology. 2015.
13. *Pryor D. V., Thistle M. R., and Shirazi N.* Text searching on Splash 2 // Proc. Workshop on FPGAs for Custom Computing Machines. 1993.
14. *Aho A. V., Hill M., Hopcroft J., Ulman J. D.* Data structures and algorithms. 1983.

15. *Андрейченко Л. Н.* Русский язык. Фонетика и фонология. Орфоэпия. Графика и орфография / Под ред. *Г. Г. Инфантовой и Н. А. Сениной*. М.: Флинта, 2003.

16. *Malyshev B., Samarin A., and Vulis D.* Russian T&jX // TUGboat. 1991. Vol. 12, N 2. P. 212—214.

17. *Зализняк А. А*. Грамматический словарь русского языка. М.: АСТ-Пресс, 2008.

18. *Грановский Д. В., Бочаров В. В., Бичинева С. В.* Открытый корпус: принципы работы и перспективы // Тр. науч. семинара „Компьютерная лингвистика и развитие семантического поиска в Интернете“ XIII Всерос. объедин. конф. „Интернет и современное общество“. Санкт-Петербург, 19—22 октября 2010. 94 с.

19. *Жуков М. А., Афанасьев Д. Б.* Анализ и оценка минимального уровня префиксного дерева в системе бесхешевой дедупликации // Научно-технический вестник информационных технологий, механики и оптики. 2015. Т. 15, № 3. С. 470—475.

20. *Cisłak A. and Grabowski S.* A practical index for approximate dictionary matching with few mismatches // arXiv:1501.04948. 2015.

Сведения об авторах

Александр Владимирович Покид — ООО „СВД Встраиваемые Системы“; инженер-программист; E-mail: a.pokid@mail.ru

Сергей Викторович Клименков — Университет ИТМО; кафедра вычислительной техники; ассистент; E-mail: serge@cs.ifmo.ru

Евгений Алексеевич Цопа — Университет ИТМО; кафедра вычислительной техники; ассистент; E-mail: jek@cs.ifmo.ru

Сергей Александрович Жмылёв — ООО „Сервиком“, отдел технического обеспечения; инженер; E-mail: korg@tune-it.ru

Нино Мерабиевна Ткешелашвили — ООО „Элком“, отдел разработки программного обеспечения; программист; E-mail: ninomt@elcom.spb.ru

Рекомендована кафедрой вычислительной техники

Поступила в редакцию 03.07.17 г.

Ссылка для цитирования: *Покид А. В., Клименков С. В., Цопа Е. А., Жмылёв С. А., Ткешелашвили Н. М.* Метод быстрого поиска узлов семантической сети по точному совпадению словоформы // Изв. вузов. Приборостроение. 2017. Т. 60, № 10. С. 932—939.

QUICK SEARCH METHOD FOR NODES OF A SEMANTIC NETWORK BY EXACT WORD FORMS MATCHING

A. V. Pokid[1], S. V. Klimenkov[2], E. A. Tsopa[2], S. A. Zhmylev[3], N. M. Tkeshelashvili[4]

[1]*SVD Embedded Systems Ltd.,196128, St. Petersburg, Russia*

[2]*ITMO University, 197101, St. Petersburg, Russia*

[3]*Servicom Ltd., 195197, St. Petersburg, Russia*

[4]*Elcom Ltd., 195197, St. Petersburg, Russia*

Development and usage of ontologies is an important part of modern text analysis application. When huge amount of text is analyzed, the lookup time in ontology becomes critical bottleneck. An approach to creation of search prefix tree of wordforms' parts called *x*-gram is proposed. Division of the wordform into 3- and 4-gramms as well as phonetic and morphological syllable for Russian language is used. Every *x*-gram is represented by numerical index that allows its storage in the plain array. Resulting arrays are very sparse, so approach uses compactification to "insert" one array into another. When looking for a word, it is split into *x*-grams, the index for every x-gram is computed, consequent lookup is performed in constructed arrays, where each array corresponds to a single level of prefix tree. The developed program demonstrates the advantage of 36—50 % over Google dense hash-map in seek time and 12 % over Google sparse hash-map in memory consumption for set of Russian wordforms extracted from well-known grammatical dictionary and Russian National Corpus. This approach is well suited for dictionary search in rarely changing wordform sets, such as ontology based on Russian Wiktionary.

Keywords: dictionary lookup, ontology based on Wiktionary, prefix tree, syllable, *n*-gramm, hash-tables, exact match

Data on authors

Alexander V. Pokid	—	SVD Embedded Systems Ltd.; Engineer-Programmer; E-mail: a.pokid@mail.ru
Sergey V. Klimenkov	—	ITMO University, Department of Computer Science; Assistant; E-mail: serge@cs.ifmo.ru
Evgeny A. Tsopa	—	ITMO University, Department of Computer Science; Assistant; E-mail: jek@cs.ifmo.ru
Sergey A. Zhmylev	—	Servicom Ltd., Technical Supply Department; Engineer; E-mail: korg@tune-it.ru
Nino M. Tkeshelashvili	—	Elcom Ltd., Software Development Department; Programmer; E-mail: ninomt@elcom.spb.ru

For citation: Pokid A. V., Klimenkov S. V., Tsopa E. A., Zhmylev S. A., Tkeshelashvili N. M. Quick search method for nodes of a semantic network by exact word forms matching. *Journal of Instrument Engineering*. 2017. Vol. 60, N 10. P. 932—939 (in Russian).

DOI: 10.17586/0021-3454- 2017-60-10-932-939