КОМПЬЮТЕРНАЯ ЛИНГВИСТИКА И ОБРАБОТКА ЕСТЕСТВЕННОГО ЯЗЫКА

УДК 004.021

СРАВНЕНИЕ АЛГОРИТМОВ ПОСТРОЕНИЯ ПОИСКОВОГО ИНДЕКСА

М. А. Артемов, Г. Г. Бердзенишвили

Воронежский государственный университет

Поступила в редакцию 28.04.2014 г.

Аннотация. Задача поиска ближайшего соседа встречается во множестве областей, таких как распознавание образов, классификация текстов, исправление ошибок и др. Для решения задачи поиска похожих строк (ближайшего соседа) с заданной точностью используются различные способы построения индекса. Одним из таких индексов является индекс, построенный на основе BK-дерева. Данный способ построения индекса позволяет решать вышеуказанную задачу достаточно эффективно. Однако, алгоритм требует сложных вычислений, занимающих много времени и требующих большое количество ресурсов. В данной статье представлено описание алгоритма HEngine, позволяющего построить более эффективный индекс, а также сравнение с индексом, построенным на основе BK-дерева, как по скорости поиска, так и по используемым ресурсам.
Ключевые слова: BK-Tree, HEngine, поиск по расстоянию Хэмминга.
Annotation. The task of finding the nearest neighbor is applicable in many areas such as image recognition, text classification, error correction, etc. There are various ways of constructing the index to solve the problem of searching for similar strings (nearest neighbor) with a given accuracy. One such index is the index based on the BK-tree. This method of constructing the index allows you to do this quite effectively. However, the algorithm requires complex calculations, which are time-consuming and requires many resources. This article presents the description of the algorithm HEngine, which allows you to build a better index, and comparison with an index based on the BK-tree on the speed of the search and the used resources.
Keywords: BK-Tree, HEngine, Hamming distance query processing.

ВВЕДЕНИЕ

Задача поиска ближайшего соседа заключается в отыскании среди множества элементов, расположенных в метрическом пространстве, элементов близких к заданному, согласно некоторой функции близости (метрики).

Примером такой задачи является задача поиска изображения по видео. Видео разбивается на набор изображений, которые в дальнейшем преобразуются в бинарные строки. В полученном наборе строк происходит поиск запрашиваемой строки (другого изображения, преобразованного в строку таким же образом) с заданной степенью похожести. Для сравнения двух строк используется расстояние Хэмминга.

Для задачи поиска похожих строк (ближайшего соседа) с заданной точностью используются различные способы построения индекса и алгоритмы.

Похожие задачи рассматриваются в работах [1], [2], в которых предлагается метод поиска оптимальной ER-модели, основанный на поиске оптимального пути по графу или с помощью генетического алгоритма.

В работе [3] рассматривается метод разбиения веб-страниц на семантические блоки с целью выявления схожих документов. Для решения задачи поиска в вышеуказанной работе используется алгоритм шинглирования.

В данной работе рассмотрены два способа построения поисковых индексов: на основе BK-дерева и с помощью алгоритма HEngine. Также проведено их сравнение по скорости поиска и используемым ресурсам.

ЗАДАЧА ПОИСКА ПОХОЖИХ СТРОК

Одним из способов определения степени похожести двух строк является расстояние Хэмминга – число позиций, в которых соответствующие символы двух слов одинаковой длины различны. В более общем случае расстояние Хэмминга применяется для строк одинаковой длины любых q -ичных алфавитов и служит метрикой различия (функцией, определяющей расстояние в метрическом пространстве) объектов одинаковой размерности.

В задаче поиска похожих строк рассматривается база данных бинарных строк T размера n, где длина каждой строки m символов. На вход поступает запрашиваемая строка a и требуемое расстояние Хэмминга k. Задача сводится к поиску всех строк, которые находятся в пределах расстояния k (в статье рассмотрен случай, когда расстояние k задано заранее, т. е. статическая задача поиска).

BK-ДЕРЕВЬЯ

Деревья Буркхарда Келлера являются метрическими деревьями, алгоритмы построения таких деревьев основаны на свойстве метрики отвечать неравенству треугольника. Это свойство позволяет метрикам образовывать метрические пространства произвольной размерности. На основании этого свойства можно построить структуру данных, позволяющую осуществлять поиск в таком метрическом пространстве, которой и являются деревья Буркхарда Келлера.

Для построения дерева выбирается любая строка из набора строк и устанавливается корнем дерева, а каждая следующая строка добавляется относительно расстояния от корня дерева. У любого узла могут быть потомки только с различным расстоянием до этого узла.

Например, пусть дан набор слов: «Корень», «Голень», «Корпус», «Колдун», «Камень», «Король», «Корова», «Костюм». Необходимо построить BK-дерево. Для этого нужно взять любое слово, например, «Корень», и сделать его корневым узлом дерева. Расстояние Хэмминга между «Корень» и «Голень» равно 2, поэтому этот узел добавится с пометкой 2 на дуге. Аналогично и для слов «Корпус» и «Колдун». Эти слова также добавляются непосредственно к корню, т. к. расстояние отличается от 2. Расстояние Хэмминга между словом «Камень» и корнем дерева равно 2. Узел с таким расстоянием уже есть («Голень»), поэтому добавление слова «Камень» происходит к слову «Голень». Аналогично добавляются и остальные слова (рис. 1).

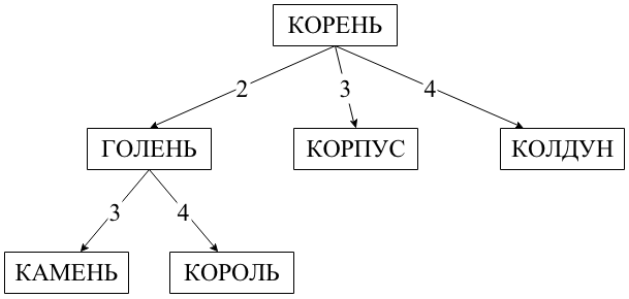

Рис. 1. Построенное BK-дерево

Для поиска всех строк, которые находятся на расстоянии N от запрашиваемой, необходимо выполнить следующую последовательность шагов:

1. Вычислить расстояние Хэмминга D между корнем и строкой.
2. Если расстояние меньше заданного, то добавить в список найденных.
3. Для всех потомков корня, у которых расстояние до родительского узла находится в пределах $[D-N, D+N]$, выполнить такие же шаги.

АЛГОРИТМ HENGINE

Существует три решения статической задачи: линейный поиск, расширение запроса и расширение базы данных. Под расшире-

нием запроса имеется в виду генерация всех возможных вариантов строк, которые вписываются в заданное расстояние для первоначальной строки. Расширение базы данных подразумевает создание множества копий этой базы данных, где также генерируются все возможные варианты, которые отвечают требованиям необходимого расстояния, либо данные обрабатываются каким-то другим способом.

HEngine [4] использует комбинацию этих трех методов для эффективного балансирования между памятью и временем исполнения.

Утверждается, что если для двух строк a и b расстояние $HD(a,b) \le k$, то если поделить строки a и b на подстроки методом *rcut*, используя фактор сегментации $r \ge \lfloor k/2 \rfloor + 1$, обязательно найдется, по крайней мере, $q = r - \lfloor k/2 \rfloor$ подстрок, когда их расстояние не будет превышать единицу, т. е. $HD(a_i, b_i) \le 1$.

Для того, чтобы выделить подстроки из базовой строки методом *rcut*, необходимо выбрать значение, названное фактором сегментации, которое удовлетворяет условию $r \ge \lfloor k/2 \rfloor + 1$. Длина первых $r - (m \bmod r)$ подстрок будет иметь длину $\lfloor m/r \rfloor$ символов, а последних $m \bmod r$ подстрок – $\lceil m/r \rceil$, где m – это длина строки, $\lfloor\ \rfloor$ – округление до ближайшего целого снизу, а $\lceil\ \rceil$ – округление до ближайшего целого сверху.

Пусть даны две бинарные строки длиной $m = 8$ бит: $A = 11110000$ и $B = 11010001$, расстояние между ними $k = 2$. Выбирается фактор сегментации $r = 2/2 + 1 = 2$, т. е. всего будет 2 подстроки длиной $m/r = 4$ бита: $a_1 = 1111$, $a_2 = 0000$, $b_1 = 1101$, $b_2 = 0001$. Если подсчитать расстояние между соответствующими подстроками, то, по крайней мере, $(q = 2 - 2/2 = 1)$ одна подстрока совпадет или их расстояние не будет превышать единицу. Действительно,

$$HD(a_1, b_1) = HD(1111, 1101) = 1 \text{ и}$$

$$HD(a_2, b_2) = HD(0000, 0001) = 1.$$

Подстроки базовой строки были названы сигнатурами. Сигнатуры или подстроки a_i и b_i называются совместимыми друг с другом, а если их количество отличающихся битов не больше единицы, то эти сигнатуры называются совпадающими.

Главная идея алгоритма HEngine – это подготовить базу данных таким образом, чтобы найти совпадающие сигнатуры и затем выбрать те строки, которые находятся в пределах требуемого расстояния Хэмминга.

Таким образом, если правильно поделить строку на подстроки, то, по крайней мере, одна подстрока совпадет с соответствующей подстрокой либо количество отличающихся битов не будет превышать единицу (сигнатуры совпадут). Это означает, что не надо проводить полный перебор по всем строкам из базы данных, а требуется сначала найти те сигнатуры, которые совпадут, т. е. подстроки будут отличаться максимум на единицу.

Для поиска совпадающих сигнатур выбран метод двоичного поиска, но он требует, чтобы список строк был отсортирован. Так как из одной строки получается несколько подстрок, то чтобы произвести двоичный поиск по списку подстрок, надо чтобы каждый такой список был отсортирован заранее. Поэтому здесь применяется метод расширения базы данных, т. е. создание нескольких таблиц, каждая для своей подстроки или сигнатуры. Такая таблица называется таблицей сигнатур, а совокупность таких таблиц – набором сигнатур.

После предварительной обработки базы данных имеется несколько копий отсортированных таблиц, каждая для своей подстроки. Для поиска подстроки необходимо из запрашиваемой строки получить сигнатуры тем же способом, который был использован при создании таблиц сигнатур. Также известно, что необходимые подстроки отличаются максимум на один элемент. Чтобы найти их, потребуется воспользоваться методом расширения запроса. Другими словами, требуется для выбранной подстроки сгенерировать все комбинации, включая саму эту подстроку, при которых различие будет максимум на один элемент. Количество таких комбинаций будет равно длине подстроки + 1. Далее необходимо произвести двоичный поиск в соответствующей таблице сигнатур на полное совпадение. Такие действия надо произвести для всех подстрок и для всех таблиц.

В финальной части алгоритма требуется отфильтровать те строки, которые не попадают в заданный предел расстояния Хэмминга, т. е. произвести линейный поиск по найденным строкам и оставить только те строки, которые отвечают условию $HD(a,b) \le k$.

На рис. 2 показан пример работы алгоритма поиска.

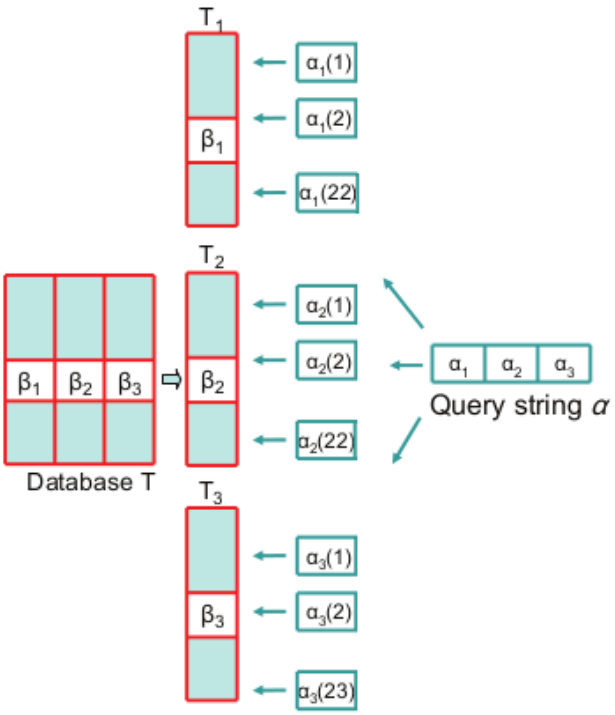

Рис. 2. Упрощенная схема обработки запросов к таблицам сигнатур

Например, для строки длиной 64 символа и предела расстояния 4, фактор сегментации равен 3, таким образом, будет только 3 подстроки на строку. T_1, T_2 и T_3 – это таблицы сигнатур, содержащие только подстроки B_1, B_2, B_3 соответственно, длиной 21, 21 и 22 бита. Запрашиваемая строка делится на подстроки. Далее для соответствующих подстрок генерируются диапазон сигнатур. Для первой и второй сигнатуры количество комбинаций будет 22. А последняя сигнатура дает 23 варианта. И в конце производится двоичный поиск.

СРАВНЕНИЕ АЛГОРИТМОВ

Для сравнения алгоритмов была реализована программа на языке программирования C++, которая позволяет выполнять поиск с помощью алгоритма HEngine, BK-дерева и линейного поиска (для наглядности сравнения).

Тест 1. Сравнение скорости поиска алгоритмов (рис. 3).

Алгоритм HEngine выполняет поиск значительно быстрее, чем BK-дерево (тест проводился для расстояния Хэмминга, равного 40, количество искомых строк равнялось 1000, для другого количества искомых строк график имеет похожий вид). Действительно, предварительные сортировки и бинарный поиск позволяют сильно сократить время на проверку строк, не удовлетворяющих заданному расстоянию Хэмминга.

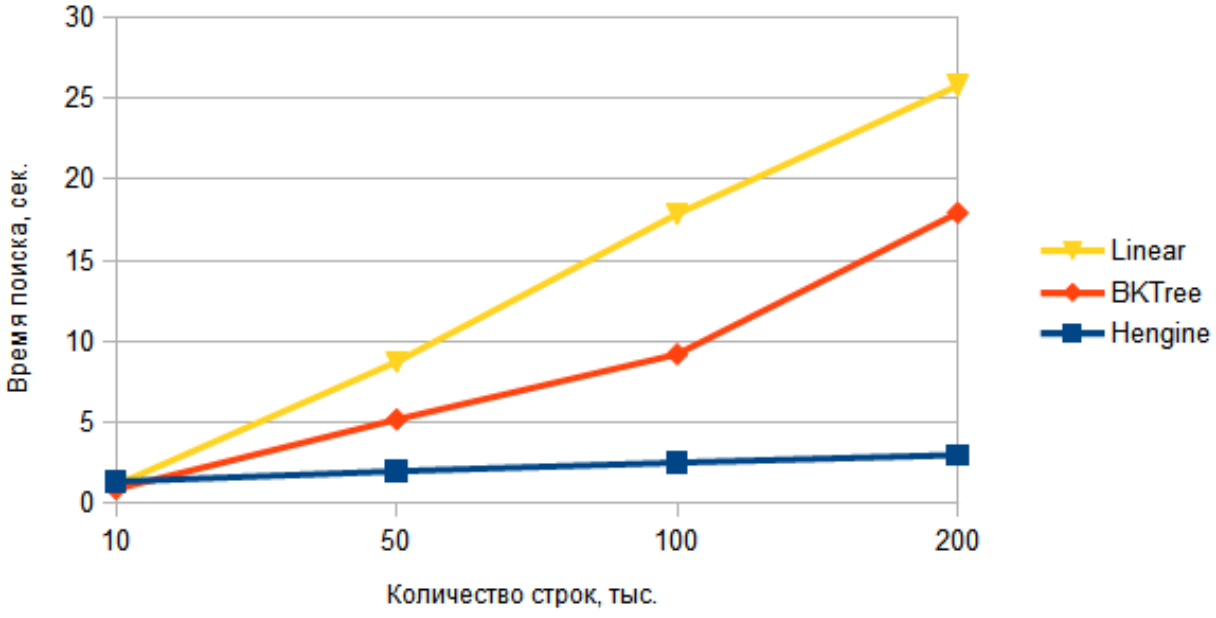

Рис. 3. Сравнение скорости поиска

Тест 2. Сравнение времени подготовки к поиску (рис. 4).

Подготовка к поиску с помощью алгоритма HEngine выполняется чуть дольше остальных алгоритмов из-за разбиений на подстроки и сортировок (тест проводился для расстояния Хэмминга, равного 40). На рис. 5 изображена диаграмма с суммарным временем подготовки и поиска для каждого из алгоритмов.

Несмотря на более длительную подготовку к поиску с помощью алгоритма HEngine, суммарное время на подготовку и поиск значительно меньше, чем для линейного поиска или для поиска с помощью BK-дерева. Это обусловлено более быстрым временем поиска по сравнению с другими алгоритмами, которое компенсирует отставание по времени подготовки.

Тест 3. Сравнение объемов занимаемой памяти индексами (рис. 6).

Индекс, построенный с помощью алгоритма HEngine, в среднем занимает в полтора раза больше памяти, чем другие алгоритмы, т. к. требует хранения дополнительной информации, такой как перестановки для искомых строк (тест проводился для расстояния Хэмминга, равного 40).

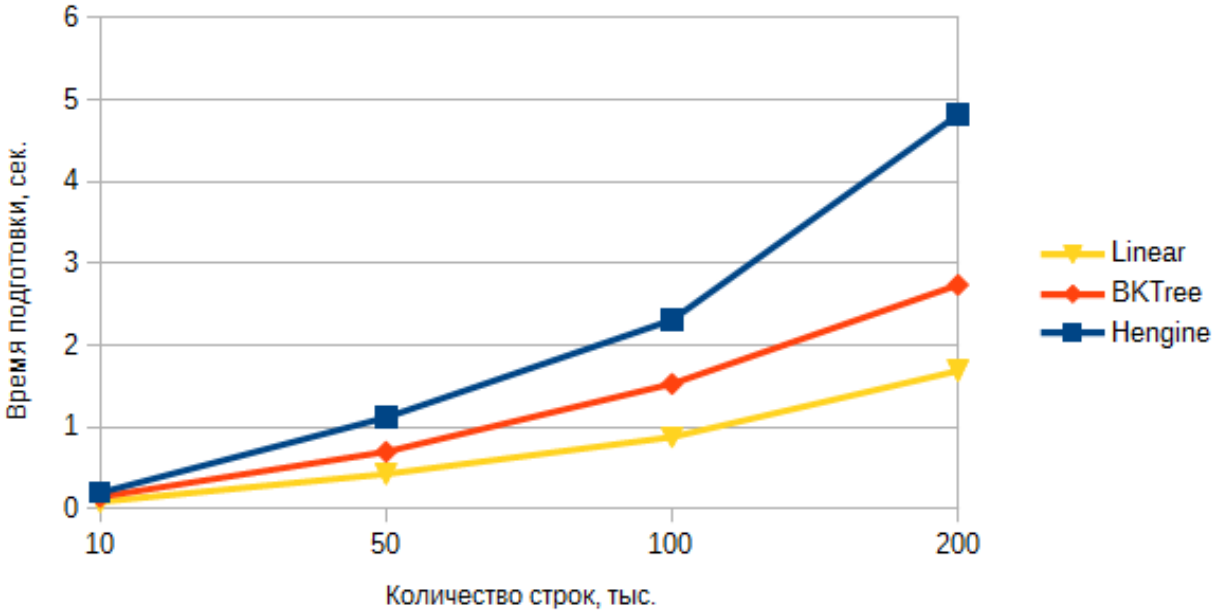

Рис. 4. Сравнение времени подготовки

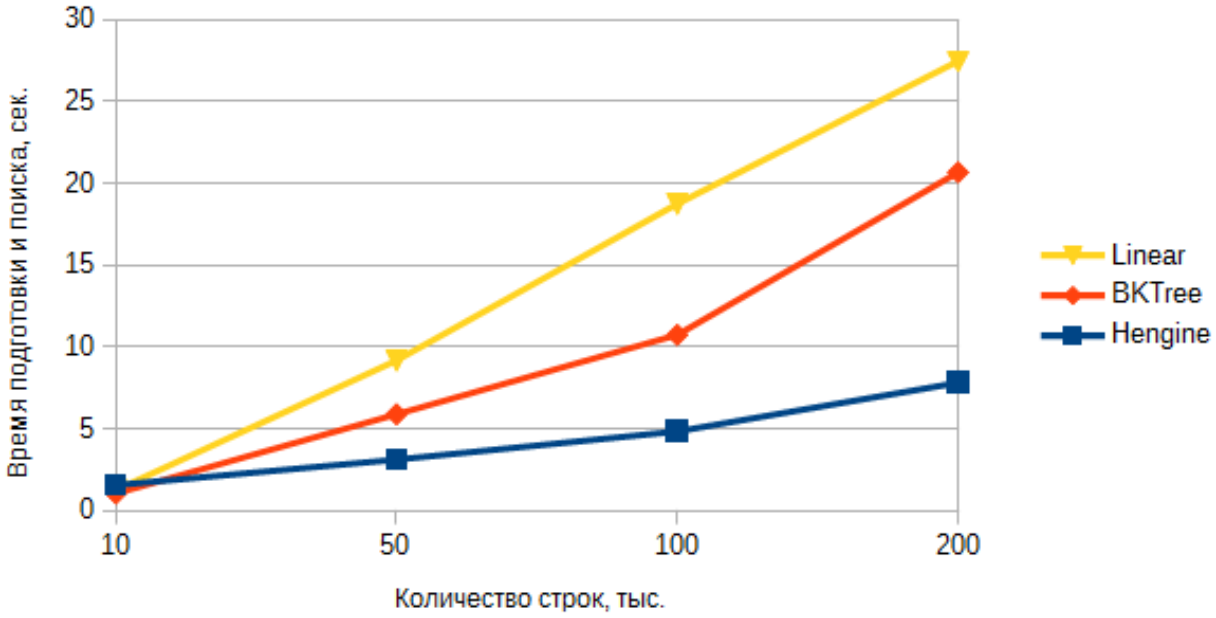

Рис. 5. Сравнение суммарного времени подготовки и поиска

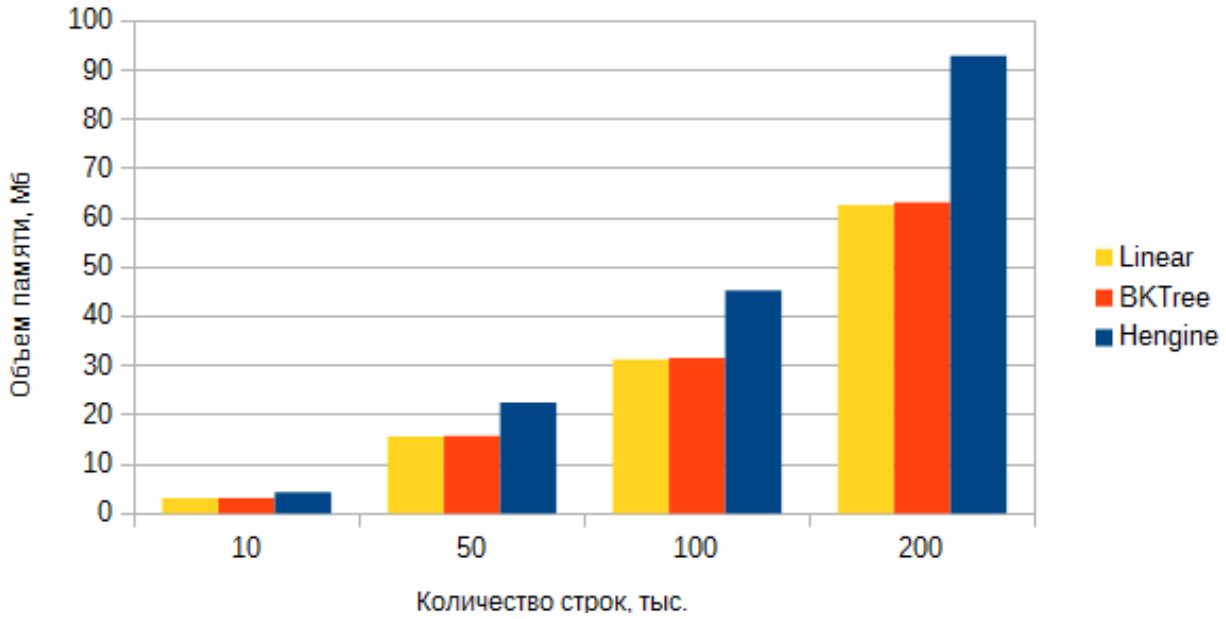

Рис. 6. Сравнение объемов занимаемой алгоритмами памяти

ЗАКЛЮЧЕНИЕ

В результате проделанной работы было проведено сравнение алгоритма HEngine с BK-деревом. Несмотря на то, что индекс, построенный с помощью алгоритма HEngine, занимает больше памяти, чем аналогичный индекс, построенный на основе BK-дерева, эффективность первого индекса значительно выше, чем второго. Это подтверждается проведенными тестами, в которых сравнивалась скорость построения и скорость поиска этих двух алгоритмов. К плюсам алгоритма HEngine также можно отнести возможность распараллеливания выполнения, что должно еще сильнее ускорить поиск, в отличие от BK-дерева, параллельный поиск с помощью которого организовать не получится.

Таким образом, индекс, построенный с помощью алгоритма HEngine, позволит значительно ускорить решение задачи поиска похожих строк.

СПИСОК ЛИТЕРАТУРЫ

1. *Тюкачев Н. А.* Выбор ER-модели для описания поверхности трехмерных тел / Н. А. Тюкачев // Вестник ВГТУ. – 2009. – Т. 5, № 4. – С. 14 24.

2. *Шалиткин А. В.* Решение обобщенной задачи поиска оптимальной ER-модели на основе генетического алгоритма / А. В. Шалиткин, Н. А. Тюкачев, П. А. Цветков // Вестник Воронежского Государственного Технического Университета. – 2011. – Т. 7, № 3. – С. 204– 207.

3. *Косинов Д. И.* Метод разбиения веб-страниц на семантические блоки с целью выявления схожих документов / Д. И. Косинов // Вестник Воронежского Государственного Университета. Серия: Системный анализ и информационные технологии. – 2008. – № 2. – С. 123–127.

4. Large Scale Hamming Distance Query Processing / A. X. Liu, K. Shen, E. Torng // The IEEE International Conference on Data Engineering. – Department of Computer Science and Engineering Michigan State University East Lansing, MI 48824, U.S.A., 2011. – С. 553–564.

Артемов М. А. – д.ф.-м.н., Воронежский государственный университет, ф-т ПММ, заведующий кафедрой Программного обеспечения и администрирования информационных систем. Тел.: (473) 220-83-37
E-mail: Artemov_m_a@mail.ru

Бердзенишвили Г. Г. – аспирант, Воронежский государственный университет, ф-т ПММ, кафедра Программного обеспечения и администрирования информационных систем. Тел.: 952-558-96-03
E-mail: gg.berdzenishvili@gmail.com

Artemov M. A. – Doctor of Phys. And Mathem. Science, Professor, Voronezh State University, AMM Faculty, Head of the Chair of Programming Software and Administration of Information Systems. Tel.: (473) 220-83-37
E-mail: Artemov_m_a@mail.ru

Berdzenishvili G. G. – postgraduate student, Voronezh State University, AMM faculty, Chair of Programming Software and of Administration of Information Systems. Tel.: 952-558-96-03
E-mail: gg.berdzenishvili@gmail.com