УДК 004.046

А. А. Трифонов

АЛГОРИТМЫ ПОСТРОЕНИЯ ИНВЕРТИРОВАННОГО ИНДЕКСА ДЛЯ КОЛЛЕКЦИИ ТЕКСТОВЫХ ДАННЫХ

Аннотация. *Актуальность и цели.* С геометрическим ростом объемов информации в XXI в. все труднее находить и отбирать полезные и качественные данные. Налицо эффект так называемого «ухудшающего отбора» и большого количества «шума». Особенно эти явления заметны в глобальной компьютерной сети Интернет. При создании информационно-поисковых систем размеры обрабатываемых текстовых коллекций зачастую очень велики, что приводит к необходимости усовершенствования методов и средств построения поисковых систем. Для выделения качественной текстовой информации используются методы и средства поиска и анализа текстовой информации. Для того чтобы избежать последовательного просмотра текстов при выполнении каждого запроса, заранее составляется инвертированный индекс документов, который ставит в соответствие терминам те документы из коллекции, в которых они встречаются. Целью данной работы является подробный анализ существующих алгоритмов построения инвертированного индекса для текстовой коллекции, выделение их достоинств и недостатков. Кроме этого, необходимо сравнить временную сложность анализируемых алгоритмов, что позволит сделать выводы об обоснованности применения каждого из них в том или ином случае. *Материалы и методы.* При построении систем информационного поиска многие решения зависят от характеристик компьютерного обеспечения, на котором будет развернута система, поэтому способы построения индекса могут быть разделены на две категории: построение, основанное на памяти, и построение, основанное на диске. Данные проведенных исследований показывают, что производительность алгоритмов построения индекса очень зависит от количества оперативной памяти, доступной процессу индексации. Учитывая специфику алгоритмов индексирования, сравнивать их сложность имеет смысл, когда объем коллекции больше или меньше объема M оперативной памяти ПК, на котором выполняется индексирование. *Результаты.* Исследована временная сложность анализируемых алгоритмов построения инвертированного индекса для текстовой коллекции в зависимости от различных параметров. Инвертированный индекс обычно является слишком большим, чтобы быть загруженным полностью в оперативную память. Если объем оперативной памяти, доступный процессу индексации, является слишком маленьким, чтобы позволить индексу быть созданным полностью в оперативной памяти, то описанный способ построения индекса в памяти может быть расширен до основанного на слиянии метода, в котором текстовый набор динамически делится на поднаборы, исходя из доступного количества оперативной памяти. Проведено сравнение временной сложности анализируемых алгоритмов в зависимости от объема оперативной памяти ПК, на котором выполняется индексирование, что позволяет сделать выводы об обоснованности применения каждого из них в том или ином случае. *Выводы.* Построение индекса, основанное на сортировке, стоит применять, когда объем текстовой коллекции не превышает несколько гигабайт. При увеличении объема коллекции более эффективным становится построение индекса, основанное на слиянии.

Ключевые слова: информационный поиск, коллекция документов, инвертированный индекс, построение индекса.

A. A. Trifonov

ALGORITHMS OF INVERTED INDEX CONSTRUCTION FOR TEXT DATA COLLECTION

Abstract. *Background.* With the rapid growth of the information in the in the XXI century it has become increasingly difficult to find and select useful and qualitative data. This is the so called "adverse selection" effect and a lot of "noise." These phenomena are especially visible in the global computer network. When creating information retrieval systems the sizes of the processed text collections are often very high and that demands improved methods and tools for building search engines. Methods and tools for data analysis of textual information are used for selection of the high quality textual information. So that to avoid a sequential survey of the texts when fulfilling each request, the inverted index of documents, which assigns to the terms of those documents from the collection, in which they occur is compiled in advance. The purpose of this paper is a detailed analysis of the existing algorithms for an inverted index construction for the text collection, their advantages and disadvantages. In addition, it is necessary to compare the time complexity of the analyzed algorithms, which allows to draw conclusions about the validity of each of them in a particular case. *Materials and methods.* In information retrieval systems, the solutions depend on the characteristics of computer software, which system will be deployed. Index defining methods can be divided into two categories: the first group is based on the memory, and the second is based on the disc. Research data show that the performance of algorithms for the index construction is highly dependent on the amount of memory available to the process of indexing. Taking into account the specificity of the indexing algorithm, it makes sense to compare their complexity when the volume of the collection is greater than or less than the amount M of RAM PC running the indexing. *Results.* The time complexity of the analyzed algorithms for an inverted index construction for a text collection based on various parameters has been investigated. An inverted index is usually too large to be completely loaded into the memory. If the amount of memory available to the process of indexing is too small to allow the index to be created entirely in memory, the described method of constructing the index in memory can be expanded up to the merge-based method in which a set of dynamic text is divided into subsets based on the amount of RAM available. Comparison of the time complexity of the algorithms depending on the amount of memory in the PC that is running the indexing, allows to draw conclusions about the validity of each of them in a particular case. *Conclusions.* Sort-based index construction should be applied when the amount of text collection is not more than a few gigabytes. When the volume of the collection is increasing the construction of the index, based on the merger seems more efficient.

Key words: information retrieval, text collection, inverted index, index construction.

Введение

С геометрическим ростом объемов информации в XXI в. все труднее находить и отбирать полезные и качественные данные. Налицо эффект так называемого «ухудшающего отбора» и большого количества «шума». Особенно эти явления заметны в глобальной компьютерной сети Интернет. Для выделения качественной текстовой информации используются методы и средства поиска и анализа текстовой информации.

Чаще всего под информационным поиском понимается процесс поиска в большой коллекции (хранящейся, как правило, в памяти компьютеров) не-

коего неструктурированного материала (обычно документа), удовлетворяющего информационные потребности [1]. Прямой поиск по тексту может быть очень эффективным, особенно на современных компьютерах, тем не менее во многих случаях необходимо нечто большее:

– иногда нужно быстро обработать большую коллекцию документов, которая составляет миллиарды и триллионы слов (для сравнения, общий объем тома собрания избранных произведений Шекспира составляет около одного миллиона слов);

– иногда необходимо выполнить ранжированный поиск. Во многих ситуациях нам нужен не просто ответ, а наилучший ответ на запрос относительно документов, содержащих определенное слово.

Для того чтобы избежать последовательного просмотра текстов при выполнении каждого запроса, заранее составляется инвертированный индекс документов, который ставит в соответствие терминам те документы из коллекции, в которых они встречаются. С точки зрения системы информационного поиска инвертированный индекс – структура данных, в которой для каждого слова коллекции документов в соответствующем списке перечислены все места в коллекции документов, в которых оно встретилось [2]. Инвертированный индекс – одна из важнейших концепций информационного поиска. Основная идея инвертированного индекса проиллюстрирована на рис. 1. Сначала в память записывается словарь терминов. Затем для каждого термина создается список, в котором указаны документы, содержащие данный термин. Каждый элемент списка, содержащий информацию о том, что термин находится в документе, называется словопозицией. Соответствующий список называется списком словопозиций. Словарь, представленный на рис. 1, выстроен в алфавитном порядке, а каждый список словопозиций упорядочен по идентификаторам документов.

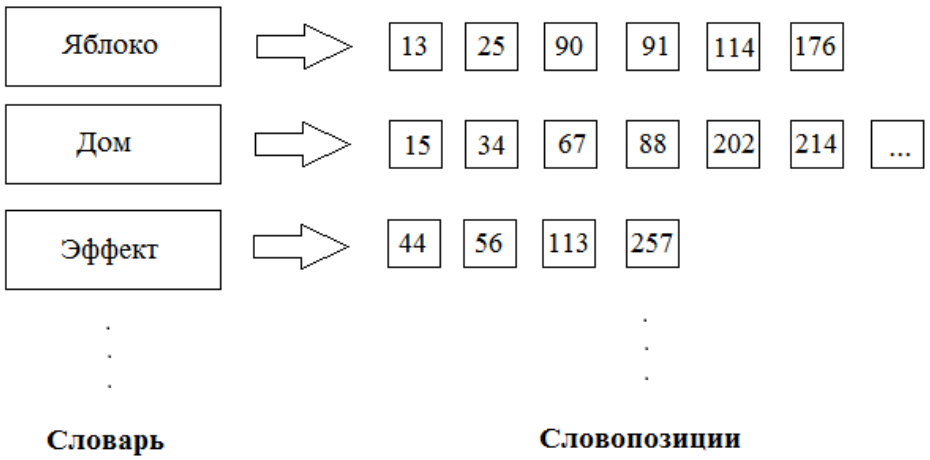

Рис. 1. Две части инвертированного индекса. Словарь обычно находится в памяти вместе с указателями на каждый список словопозиций, которые хранятся на диске

При построении систем информационного поиска многие решения зависят от характеристик компьютерного обеспечения, на котором будет развернута система, поэтому способы построения индекса могут быть разделены

на две категории: построение, основанное на памяти, и построение, основанное на диске [3]. Построение индекса, основанное на памяти, создает индекс для данного текстового набора полностью в оперативной памяти и поэтому может использоваться только, если набор небольшой относительно объема свободной оперативной памяти. Однако эти способы формируют основу для более сложных, основанных на диске способов построения индекса. Основанные на диске методы могут использоваться, чтобы создать индексы для очень больших наборов, намного больше, чем доступное количество оперативной памяти.

1. Индексирование, основанное на сортировке

Для повышения эффективности индексирования представим термины в виде идентификаторов терминов termID, где каждый идентификатор termID представляет собой уникальный порядковый номер. Если объема оперативной памяти недостаточно, необходим алгоритм внешней сортировки, т.е. алгоритм, использующий диск. Для того чтобы достичь приемлемой скорости, такой алгоритм должен минимизировать количество случайных перемещений головки по диску во время сортировки – последовательное считывание данных выполняется намного быстрее. Одним из решений этой проблемы является алгоритм блочного индексирования, основанного на сортировке (blocked sort-based indexing), или BSBI (рис. 2). Алгоритм BSBI сегментирует коллекцию на равные части, сортирует пары «termID-docID» каждой части в памяти, сохраняет промежуточные отсортированные результаты на диске и объединяет все промежуточные результаты в окончательный индекс.

```
BSBIndexConstruction ()
1    n ← 0
2    while (не просмотрены все документы)
3    do n ← n+1
4         block ← ParseNextBlock ()
5         BSBI-Invert(block)
6         WriteBlockToDisk (block, f_n)
7    MergeBlocks (f_1, …, f_n, f_merged)
```

Рис. 2. Индексирование, основанное на сортировке. Этот алгоритм хранит инвертированные блоки в файлах $f_1, \ldots, f_n$, а объединенный индекс – в файле f_{merged}

Алгоритм формирует по документам пары «termID-docID» и накапливает их в памяти, пока не будет заполнен блок фиксированного размера. Размер блока подбирается так, чтобы он помещался в памяти и допускал быструю сортировку. Затем блок инвертируется и записывается на диск. Инвертирование выполняется в два этапа. Во-первых, сортируются пары «termID-docID». Во-вторых, все пары «termID-docID» с одинаковыми идентификаторами termID формируют инвертированный список, а в качестве словопозиции выступает только идентификатор документа docID. Результат, представляющий собой инвертированный индекс для считанного блока, записывается на диск. Применяя этот алгоритм к текстовой коллекции и предполагая, что

в оперативной памяти можно разместить 10 миллионов пар «termID-docID», получаем десять блоков, каждый из которых является обратным индексом для одной из частей коллекции. На заключительном этапе алгоритм одновременно выполняет слияние десяти блоков в один большой объединенный индекс. Временная сложность данного алгоритма равна $\Theta(T \log T)$, так как самым сложным этапом является сортировка, а количество сортируемых элементов ограничено числом T (количеством пар «termID-docID»)

2. Индексирование, основанное на памяти

Блочное индексирование, основанное на сортировке, обладает превосходными возможностями для масштабирования, но для его реализации каждому термину необходимо поставить в соответствие идентификатор termID. Для очень больших коллекций необходимая структура данных может не поместиться в памяти [1]. Более эффективным с точки зрения масштабирования является однопроходное индексирование в памяти (single-pass in-memory indexing), или алгоритм SPIMI (рис. 3). Этот алгоритм использует термины, а не их идентификаторы, записывает словарь каждого блока на диск, а затем для нового блока начинает создавать новый словарь. При наличии достаточного объема памяти на диске с помощью алгоритма SPIMI можно проиндексировать коллекцию любого размера.

```
SPIMI-Invert (token_stream)
1    output_file ← NewFile()
2    dictionary ← NewHash()
3    while (есть свободная память)
4    do token ← next(token_stream)
5          if term(token) ∉ dictionary
6                then postings_list = AddToDictionary(dictionary, term(token))
7                else postings_list = GetPostingsList(dictionary, term(token))
8          if full(postings_list)
9                then postings_list = DoublePostingsList(dictionary, term(token))
10         AddToPostingsList (postings_list, docID(token))
11   sorted_terms ← SortTerms (dictionary)
12   WriteBlockToDisk (sorted_terms, dictionary, output_file)
13   return output_file
```

Рис. 3. Инверсия блока с помощью однопроходного индексирования в оперативной памяти

В данном контексте пары «termID-docID» называются лексемами. Лексемы обрабатываются по очереди. Когда термин встречается впервые, он добавляется в словарь и создается новый инвертированный список, т.е. этот инвертированный список возвращается для последующих появлений данного термина.

Разница между алгоритмами BSBI и SPIMI заключается в том, что алгоритм SPIMI добавляет записи непосредственно в инвертированный список. Вместо того чтобы коллекционировать все пары «termID-docID», а затем их сортировать (как это делает алгоритм BSBI), в алгоритме SPIMI каждый ин-

вертированный список является динамическим (т.е. его размер по мере надобности возрастает) и немедленно становится доступным для хранения записей. У такого подхода есть два преимущества: он быстрее, поскольку не подразумевает сортировки, и экономит память, поскольку отслеживает связь термина с соответствующим инвертированным списком. И хранить идентификаторы termID для этих списков нет необходимости. В результате блоки обрабатываемых данных могут быть намного больше, а процесс составления индекса в целом становится более эффективным. Временная сложность алгоритма SPIMI равна $\Theta(T)$, поскольку сортировка лексем не нужна и сложность всех операций не более чем линейно зависит от размера коллекции.

3. Индексирование, основанное на слиянии

В отличие от основанного на сортировке метода построения индекса, метод, основанный на слиянии, не должен поддерживать глобальные структуры данных. В частности, нет никакой потребности в глобальных уникальных идентификаторах терминов. Поэтому размер текстового набора, который будет проиндексирован, ограничивается только объемом дискового пространства, доступного, чтобы хранить временные данные и заключительный индекс, но не количеством оперативной памяти, доступной процессу индексации.

Основанная на слиянии индексация – обобщение способа построения индекса в памяти, при котором создаются инвертированные списки посредством поиска по хэш-таблице. Фактически, если набор, для которого создается индекс, является достаточно маленьким, чтобы индекс полностью помещался в оперативной памяти, тогда основанная на слиянии индексация ведет себя точно как же, как индексация в памяти, а его временная сложность равна $\Theta(T \log T)$. Однако если текстовый набор является слишком большим, чтобы индексироваться полностью в оперативной памяти, то процесс индексации выполняет динамическое разделение набора. Таким образом, этот процесс начинает создавать индекс в памяти. Как только он исчерпывает память (или когда достигается предопределенный порог использования памяти), он создает дисковый инвертированный файл, передавая индексные данные на дисковую память, удаляет индекс в памяти и продолжает индексировать. Эта процедура повторяется, пока весь набор не проиндексируется. Таким образом, временная сложность алгоритма уменьшается до $\Theta(T)$.

Результатом процесса, обрисованного в общих чертах выше, является ряд инвертированных файлов, каждый из которых представляет определенную часть целого набора. Каждый такой подындекс называется индексным разделом. На заключительном шаге индексные разделы объединяются в заключительный индекс, представляя весь текстовый набор. Списки словопозиций в индексных разделах (и в заключительном индексе) обычно сохранены в сжатой форме.

Индексные разделы, записанные на диск как промежуточные результаты процесса индексации, абсолютно независимы друг от друга. Например, нет никакой потребности в глобальных уникальных идентификаторах терминов; нет даже потребности в числовых идентификаторах терминов. Каждый термин является своим собственным идентификатором; списки словопозиций в каждом разделе сохранены в алфавитном порядке их терминов. Из-за алфавитного упорядочивания и отсутствия идентификаторов терминов слияние

отдельных разделов в заключительный индекс является прямым. Если число индексных разделов является большим (больше, чем 10), то алгоритм может быть улучшен посредством расположения индексных разделов в приоритетной очереди (например, в «куче»), упорядоченной согласно следующему термину в соответствующем разделе.

Полное время, необходимое, чтобы создать индекс для текстовой коллекции GOV2 (426 Гб текста) с помощью слияния, составляет приблизительно 4 часа [2]. Время, требуемое для выполнения заключительной операции слияния, составляет приблизительно 30 % от времени, которое требуется, чтобы сгенерировать разделы.

Алгоритм очень масштабируем: на компьютере средней мощности индексация целого набора GOV2 требует в 11 раз больше времени, чем индексация 10 % поднабора (43 Гб текста).

Есть, однако, некоторые пределы масштабируемости метода. Объединяя индексные разделы в конце процесса индексации, важно иметь по крайней мере буфер предварительного чтения умеренного размера, несколько сотен килобайтов для каждого раздела. Это помогает сохранить число поисков на диске (переходы вперед и назад между различными разделами) небольшим. Естественно, размер буфера предварительного чтения для каждого раздела ограничивается сверху числом M/n, где M – количество доступной памяти, n – число разделов. Таким образом, если n становится слишком большим, слияние становится медленным.

Сокращение объема памяти, доступного процессу индексации, имеет два эффекта. Во-первых, оно уменьшает общий объем памяти, доступный для буферов предварительного чтения. Во-вторых, оно увеличивает число индексных разделов. Таким образом, сокращение оперативной памяти на 50 % уменьшает размер каждого индексного буфера предварительного чтения на 75 %. Например, установка предела памяти M 128 Мб приводит к 3032 разделам, которые должны быть объединены, оставляя каждому разделу с буфером предварительного чтения только 43 Кб. Данные показывают, что производительность заключительной операции слияния очень зависит от количества оперативной памяти, доступной процессу индексации. При 128 Мб доступной оперативной памяти заключительное слияние длится в 6 раз дольше, чем с 1 Гб [2].

Существуют две возможные контрмеры, которые могут быть предприняты, чтобы преодолеть это ограничение. Первая – необходимо заменить простое многоканальное слияние каскадной операцией слияния. Например, если должны быть объединены 1024 индексных раздела, то можно сначала выполнить 32 операции слияния, включающие 32 раздела каждый, и затем объединить получающиеся 32 новых раздела в заключительный индекс. Вторая контрмера предполагает уменьшение места, занимаемого списками словопозиций, посредством сжатой инверсии в памяти. Сжатие словопозиций на лету, как только они попадают в индекс в памяти, позволяет процессу индексации накапливать больше словопозиций, прежде чем они исчерпают память, таким образом сокращая число создаваемых на диске индексных разделов.

Несмотря на некоторые проблемы с последней операцией слияния, основанное на слиянии построение индекса позволяет нам создавать инвертированный файл для очень больших текстовых наборов даже на одном ПК. Его преимущество перед основанным на сортировке методом состоит в том, что

он не требует глобальных уникальных идентификаторов терминов. Поэтому этот метод особенно привлекателен, если число терминов словаря является очень большим. Другое большое преимущество этого алгоритма построения индекса перед основанным на сортировке методом состоит в том, что он производит в памяти индекс, способный сразу же обрабатывать запросы. Эта функция важна, когда поисковая система должна иметь дело с динамическими текстовыми наборами.

4. Сравнение временной сложности алгоритмов

Чтобы понять, в каких случаях применять тот или иной алгоритм построения инвертированного индекса для коллекции текстовых данных, сравним их временную сложность (табл. 1). Так как основной характеристикой коллекции текстовых является ее объем V (и соответственно количество обрабатываемых лексем T), то, учитывая специфику алгоритмов индексирования, сравнивать их сложность имеет смысл, когда объем коллекции больше или меньше объема M оперативной памяти ПК, на котором выполняется индексирование.

Таблица 1

Тип индексирования	Θ (при $V < M$)	Θ (при $V > M$)
Индексирование, основанное на памяти	$\Theta(T \log T)$	$\Theta(T \log T)$
Индексирование, основанное на сортировке	$\Theta(T)$	$\Theta(T)$
Индексирование, основанное на слиянии	$\Theta(T \log T)$	$\Theta(T)$

Как видно из табл. 1 и анализа алгоритмов, индексирование, основанное на сортировке, стоит применять, когда объем текстовой коллекции не превышает несколько гигабайт. При увеличении же объема коллекции более эффективным становится индексирование, основанное на слиянии.

5. Другие типы индексов

Инвертированный индекс – только один возможный тип индекса, который может использоваться в качестве части поисковой системы.

Прямой индекс является отображением списка терминов, появляющихся в документах, на идентификаторы документов. Прямой индекс дополняет инвертированный. Он обычно используется не для фактического поискового процесса, а чтобы получить информацию о распределении термина в документе во время запроса, которое требуется методами расширения запроса, и выдать результирующие сниппеты [4]. По сравнению с извлечением этой информации из необработанных текстовых файлов, у прямого индекса есть преимущество, потому что текст уже был проанализирован и соответствующие данные могут быть извлечены более эффективно.

Файлы подписи (Фалоутсос и Кристодоулакис, 1984) являются альтернативой docID-индексам. Подобный фильтру Блума (Блум, 1970), файл подписи можно использовать, чтобы получить список документов, которые могут содержать данный термин. Чтобы узнать, существует ли фактически термин в документе, нужно обратиться непосредственно к документу (или прямому индексу). Изменяя параметры файла подписи, можно обменять время

на скорость: меньший индекс приводит к большей вероятности ложных появлений и наоборот.

Суффиксные деревья (Вайнер, 1973) и суффиксные массивы (Манбер и Майерс, 1990) могут использоваться, чтобы эффективно найти все возникновения данной последовательности *n* слов в данном текстовом наборе. Суффиксные деревья – привлекательные структуры данных для поиска фразы или поиска регулярного выражения, но они обычно больше, чем инвертированные индексы, и обеспечивают менее эффективные поисковые операции, когда хранятся на диске вместо оперативной памяти.

Заключение

Рассмотрены важные алгоритмы и структуры данных, необходимые для построения инвертированного индекса и доступа к нему. Инвертированный индекс обычно является слишком большим, чтобы быть загруженным полностью в оперативную память. Поэтому чаще всего записывают в оперативную память только словарь, который является относительно маленьким по сравнению с размером всего индекса, храня списки словопозиций на диске.

Если объем оперативной памяти, доступный процессу индексации, является слишком маленьким, чтобы позволить индексу быть созданным полностью в оперативной памяти, то способ построения индекса в памяти может быть расширен до основанного на слиянии метода, в котором текстовый набор динамически делится на поднаборы, основываясь на доступном количестве оперативной памяти. При использовании метода индексации в памяти индекс создается для каждого поднабора. После того как весь набор был проиндексирован, индексы для отдельных поднаборов объединяются в единственной многоканальной операции слияния, или каскадном процессе. Производительность основанного на слиянии построения индекса линейно зависит от размера набора. Однако заключительная операция слияния может пострадать, только если слишком маленький объем оперативной памяти доступен для буферов чтения подындексов.

Список литературы

1. **Маннинг, К.** Введение в информационный поиск : пер. с англ. / К. Маннинг, П. Рагхаван, Х. Шютце. – М. : Вильямс, 2011. – 528 с.
2. **Büttcher, S.** Cormack. Information Retrieval: Implementing and Evaluating Search Engines / Stefan Büttcher, Charles L. A. Clarke, and Gordon V. – MIT Press, 2010. – 606 с.
3. **Ландэ, Д. В.** Интернетика. Навигация в сложных сетях. Модели и алгоритмы / Д. В. Ландэ, А. А. Снарский, И. В. Безсуднов. – М. : Либроком, 2009. – 264 с.
4. **Леонтьева, Н. Н.** Автоматическое понимание текстов: системы, модели, ресурсы : учеб. пособие для студ. лингв. фак. вузов / Н. Н. Леонтьева. – М. : Академия, 2006. – 304 с.

References

1. Manning K., Ragkhavan P., Shyuttse Kh. *Vvedenie v informatsionnyy poisk: per. s angl.* [Introduction into data search: translation from English]. Moscow: Vil'yams, 2011, 528 p.
2. Büttcher, S. Charles L. A. Clarke, and Gordon V. *Cormack. Information Retrieval: Implementing and Evaluating Search Engines*. MIT Press, 2010, 606 p.

3. Lande D. V., Snarskiy A. A., Bezsudnov I. V. *Internetika. Navigatsiya v slozhnykh setyakh. Modeli i algoritmy* [Internetics. Navigation in complex networks. Models and algorithms]. Moscow: Librokom, 2009, 264 p.
4. Leont'eva N. N. *Avtomaticheskoe ponimanie tekstov: sistemy, modeli, resursy: ucheb. posobie dlya stud. lingv. fak. vuzov* [Automatic text recognition: sistems, models, resources: tutorial of linguistic faculty students]. Moscow: Akademiya, 2006, 304 p.

Трифонов Александр Александрович
аспирант, Пензенский государственный университет (Россия, г. Пенза, ул. Красная, 40)

Trifonov Aleksandr Aleksandrovich
Postgraduate student, Penza State University (40 Krasnaya street, Penza, Russia)

E-mail: alexander.a.trifonov@gmail.com

УДК 004.046

Трифонов, А. А.

Алгоритмы построения инвертированного индекса для коллекции текстовых данных / А. А. Трифонов // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2013. – № 3 (27). – С. 52–61.