THE DICTIONARY STRUCTURE FOR EFFECTIVE WORD SEARCH

WALDEMAR KARWOWSKI, PIOTR WRZECIONO

Department of Informatics, Warsaw University of Life Sciences (SGGW)

In the paper some issues connected with indexing documents in the Polish language are discussed. Algorithms for stemming and part of speech tagging, important in the process of text analysis and indexing are shortly described. Next their suitability to the Polish language, which has a very extensive inflection, is discussed. The usefulness for stemming and part of speech tagging of large dictionaries with inflected forms, like WordNet and open-source dictionary of Polish language is also described. Two dictionary structures enabling effective word searching are presented. In the final part, some tests of implemented two dictionary structures are described. Tests were made on the six actual and three crafted artificial texts. At the end conclusions of performed tests are formulated.

Keywords: Text Indexing, Text Searching, Trie

1. Introduction

Natural language processing is one of the oldest subjects of computer science. The language in the written form saved as a digital document is a good material for automatic text processing. One of the tasks of automatic text processing is indexing of documents. Growing data resources, particularly on the Internet, resulted that need for automatic indexing has grown. Indexation process is the first stage of the search support in a given context and is the basis of information retrieval. The main goal of an information retrieval system can thus be defined as "finding material (usually documents) that satisfies an information need from within large collections
(usually stored on computers)" [1]. After indexing Information Retrieval Systems can select and rank documents according to users’ queries. To start the text analysis some initial processing is necessary. Inflexion and various forms of words are the first problem that must be defeated. The main issue related to indexing is to reduce inflectional and sometimes derivationally related forms of a word to a common base form. It can be solved with stemming the most important technique used in this case. There are many stemming algorithms available, best known: the Lovins stemmer [2], Paice/Husk stemmer [3] and the Porter stemmer [4, 5]. The literature review about stemming can be found in the second chapter of work [1]. Most of the stemming methods are good in the English language. There were also attempting to adapt these methods to the East European languages [6] or Polish language [7]. Another important issue during indexing is to identify the part of speech. Part of speech tagging is the process of classifying word in a text as corresponding to a particular part of speech nouns, verbs, etc. Issues connected with this technique are described in [8]. Nowadays in English texts part of speech tagging is fairly accurate [9], many English language taggers are listed at [10]. Some methods were implemented for the Polish language [11].

To simplify stemming and part of speech tagging processes, we can use large dictionaries. In the English language such frequently used dictionary is WordNet [12]. WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. The synsets are interlinked by means of conceptual-semantic and lexical relations. The easiest way to stem by brute force, is to use a dictionary, like WordNet, which lists all words together with their stems [13]. WordNet is useful for part-of-speech tagging but one problem appears. In English a large percentage of word-forms are ambiguous. For example if we have sentences “plants need light” and “he plants flowers” we have to recognize is “plants” noun or verb? This means that in addition to the data from the WordNet it is necessary to take account of the context.

2. Dictionary structure for effective word search

In languages which have a very extensive inflection, stemming is not an easy task. The Polish language is one of them. In this language, the following parts of speech inflect: verbs, nouns, numerals, adjectives, and pronouns. The number of forms is also very large. Some verbs have about a hundred different inflectional forms [14] created on the basis of the infinitive. In the paper design of a dictionary structure to store words and their inflected forms is presented, and examining its search efficiency is discussed. This structure is part of a larger system for the automatic indexing system for agricultural texts in Polish on the semantic described in [15-17]. In the system, indexer works in two steps. During the first step, a vector
of words in the base form with the part of speech tags is prepared. In the second
step semantic tagging according to AGROVOC [18]. AGROVOC is a thesaurus in
SKOS format, covering food, nutrition, agriculture, fisheries, forestry, environment
and other areas of interest of the Food and Agriculture Organization of the United
Nations. It consists of over 32,000 concepts connected with semantic relations and
is available in 27 languages, among them is Polish. The process of semantic tag-
gging was described in [17]. In this paper we concentrate on the first step i.e. prepar-
ing vector of words in the base form, using the prepared dictionary structure.

In the beginning, it was necessary to choose the appropriate source dictionary.
WordNet was originally prepared for the English language. Currently different ver-
sions of WordNet in several languages were created. Among them Polish version,
called plWordNet, was developed at Wrocław University of Technology [19].
However, we had not the appropriate tools that enable easy integration plWordNet
with our system. Finally it was decided that indexing system in Polish will be
based on the database of words with inflected forms from an open-source dictio-
ary of Polish language [14]. This dictionary is an open project licensed under the
GPL and CC SA. It is continuously updated and contains more than 200000 rec-
ords. Moreover it is possible to download the text file with one of the several types
of dictionary, such as a spelling dictionary or inflection dictionary. In our pro-
ject we decided to use a text file with inflection dictionary to create a structure con-
taining the whole inflection dictionary together with part of speech tags.

The text file with inflection dictionary consists of lines in the form: base form,
a list of inflected forms, for example for potato we have: “ziemniak, ziemniaka,
ziemniakach, ziemniakami, ziemniakiem, ziemniakom, ziemniakowi,
ziemniaków, ziemniaku”. The method for preparing dictionary structure in memory
is following. The entire dictionary after loading is stored as an array of auxiliary
structures defined as { String baseForm; int numberOfForms; int grammaticalCat-
egory; String[] inflectionForms; }. The fields for base form, a number of forms
and inflection forms table are filled immediately. The inflection dictionary contains
only words. It does not have special tag for the parts of speech. To recognize parts
of speech, a special algorithm is applied as described in [15]. After that, the array
of auxiliary structures for all base forms is ready to use. To allow faster searching
of words there are prepared the additional array of pairs, word and index of its base
form in the array of auxiliary structures, for all words. This is obviously redundant-
y, each word is stored twice, in the inflectionForms array and array of pairs, but
this allows speed searching. The array of pairs has near four million records and is
sorted to allow the binary search. So prepared dictionary structure allows efficient
construction of vector of words in the base form with the part of speech tags of the
corresponding text. Each word from the analyzed text is searched (binary search) in
the array of pairs and then the index of the base form is found, and word is being
replaced by its base form. Of course some kinds of words, which are used only for
the building of sentence, have to be ignored. These mentioned words are conjunc-
tions, particles, pronouns or prepositions. This causes a reduction in the size of the
vector of words corresponding to the analyzed text. If in the analyzed text appears
an unknown word, i.e. not present in the dictionary, it is added to the dictionary
and remains in the resulting vector of words. Grammatical form in this case may
not be specified. Such a situation in the texts in the Polish language appears when
we have the words in a foreign language. Generally, these are the Latin, English or
other language names, for example, species of plants, animals or chemical com-
pounds, like Latin zea for maize (kukurydza in Polish). The resulting vector is in
the second step of the algorithm indexed according to terms from AGROVOC.

3. Dictionary structure based on a trie

A subsequent search of words in a sorted array containing almost four million
items takes some time. Of course on modern computers, this is an acceptable time.
However, an attempt was made to create a more efficient structure. The trie is a
kind of search tree but unlike a binary search tree, no node in the tree stores the key
associated with that node; instead, node stores only a part of key [20]. If a key is a
string trie node stores typically only one character. It is easy to see that all the de-
scendants of a node have a common prefix of the string associated with that node.

![Figure 1. Words with the common prefix in trie](image)

In the Figure 1 trie for words: z, ziąb, ziem, ziami, ziemia, ziemią, ziemniak,
ziemniaka, ziemniaki, zim, and zima, is presented. Black node means the end of the
word. We have to note that more readable would be the attachment of the labels
with the characters to the edges. In this situation, we have to add an empty start
node and to the edge between it and the current node z assign label z. This does not change the general idea of the implementation of which will be described below.

The dictionary structure has been modified using trie. The first part of implementation does not change; the array of auxiliary structures stayed the same. However, instead of the array of pairs, word and index of its base form in the array of auxiliary structures, special trie was constructed. We have to note that before entering to a trie all characters are converted to lowercase. In the inflection dictionary we have forty different characters, the Polish alphabet, space, hyphen, dot, apostrophe and é (in several borrowed phrases in French). We had to add to this set dollar sign, which overrides all other characters in addition to the listed. This is important during the searching process, if the analyzed text contains a word with another character, for example, the word in German like München, such character is replaced by $. Finally, character | was added as the end of word sign and together we have forty two characters: $, a, a, b, c, e, d, e, q, f, g, h, i, j, k, l, m, n, n, o, o, p, q, r, s, t, u, v, w, x, y, z, ż, ź, ' , ', é, |. To speed up the search, huge array of blocks having a size forty two was created. Each block represents trie node. Index of a cell corresponds to one of the forty two characters (0 for $, 1, for a, etc.), value of a cell points to proper next block. Thanks to this solution, checking each character takes constant time. In the Figure 2 sample block is presented, for simplicity, pointers to the next blocks are not typed.

```
  0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z $
```

**Figure 2.** Sample block represents trie node

Cell for the index of the end of the word is not empty. It should consist index of the basic form in the array of auxiliary structures. But in our trie a cell, indexed by ‘|’, contains an extra pointer, which indicates to a list of homonyms. This list consists of indexes of the basic forms in the array of auxiliary structures. Homonyms are not too common in the Polish language but their distinction is necessary for the proper analysis of the text. For example word mają in Polish language is inflected form from: maią (adorn something green twigs and flowers), maja (spider crab), maja (in Indian philosophy: the illusion), Maja (feminine name), Maja (a member of a group of native American peoples speaking languages of the family may), Maja (river in Russia) and mieć (to have). Of course, in a vast number of cases, the list consists of one element. In the Figure 3 fragment of the trie for words ziąb and zima is presented. In this case both words refer to only one base form. For simplicity, nodes (blocks) were shortened. As we said, the array is huge and consists of 268775010 integer cells every of size 32 bits. It means that we have 6399405 nodes in our trie \(268775010 = 6399405 \times 42\). Our application for indexing was implemented in C# language and needs about 2 GB of RAM memory.
We have to note that a trie where each node is a separate array of size 42 required even more memory. Of course in many nodes the majority of the cells contains a value of zero. However, adopted extensive solution allows for faster searching of words.

Figure 3. Fragment of trie
4. Comparing effectiveness of dictionary structures

As we said the application was implemented in C#. For binary search standard Array.BinarySearch method was used. To compare the search results of two dictionary structures we selected several publications from Agricultural Engineering Journal (Inżynieria Rolnicza - IR) the same that were used in the paper [17]. “Text A” is “Information system for acquiring data on geometry of agricultural products exemplified by a corn kernel” (Jerzy Weres: „Informatyczny system pozyskiwania danych o geometrii produktów rolniczych na przykładzie ziarniaka kukurydzy”. IR 2010 Nr 7);  “Text B” is “Assessment of the operation quality of the corn cobs and seeds processing line” (Jerzy Bieniek, Jolanta Zawada, Franciszek Molendowski, Piotr Komarnicki, Krzysztof Kwietniak: „Ocena jakości pracy linii technologicznej obróbki kolb i ziarna kukurydzy”. IR 2013 Nr 4); “Text C” is “Methodological aspects of measuring hardness of maize caryopsis” (Gabriel Czachor, Jerzy Bohdziewicz: „Metodologiczne aspekty pomiaru twardości ziarniaka kukurydzy”. IR 2013 Nr 4);  “Text D” is “Evaluation of results of irrigation applied to grain maze” (Stanisław Dudek, Jacek Źarski: „Ocena efektów zastosowania nawadniania w uprawie kukurydzy na ziarno”. IR 2005 Nr 3); “Text E” is “Comparative assessment of sugar corn grain acquisition for food purposes using cut off and threshing methods” (Mariusz Szymanek: „Ocena porównawcza pozyskiwania ziarna kukurydzy cukrowej na cele spożywcze metodą odcinania i omlotu”. IR 2009 Nr 8).

Additionally, we have prepared two “artificial” texts. “Text X” contains two thousand times word contains, this word is not present in the dictionary. “Text Y” contains two thousand times word niewybielały, this word is exactly in the center of the array of pairs for first dictionary structure and should be immediately found by the binary search algorithm. “Text Z” contains two thousand times word niewybielałych, because this word is next to word niewybielały, binary search algorithm should perform the maximum number of steps.

The results of the test are presented in Table 1 – Table 9, in the header of the table we put the number of words in the particular text. The measure is the number of processor ticks. Every test was taken three times for one loop, ten, and one hundred loops. The reason is that .NET Just In Time compiler prepares methods before the first run, if we run method next time compiled method code is in memory. The result is a certain overhead during the first loop. One hundred loops better show aspect ratio between efforts in searching in every dictionary structure. For example in Table 1 we can see that overhead for trie search is about 2300 but search itself about 75. For binary search overhead is about 4600 but search itself about 22511.
<table>
<thead>
<tr>
<th>Number of loops</th>
<th>Binary search</th>
<th>Trie search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27143</td>
<td>2451</td>
</tr>
<tr>
<td>10</td>
<td>231255</td>
<td>3062</td>
</tr>
<tr>
<td>100</td>
<td>2255729</td>
<td>9730</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of loops</th>
<th>Binary search</th>
<th>Trie search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>43324</td>
<td>2834</td>
</tr>
<tr>
<td>10</td>
<td>357828</td>
<td>3828</td>
</tr>
<tr>
<td>100</td>
<td>3496724</td>
<td>14223</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of loops</th>
<th>Binary search</th>
<th>Trie search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34909</td>
<td>2631</td>
</tr>
<tr>
<td>10</td>
<td>310671</td>
<td>3584</td>
</tr>
<tr>
<td>100</td>
<td>2997137</td>
<td>12091</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of loops</th>
<th>Binary search</th>
<th>Trie search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22365</td>
<td>1688</td>
</tr>
<tr>
<td>10</td>
<td>188085</td>
<td>2183</td>
</tr>
<tr>
<td>100</td>
<td>1820936</td>
<td>7533</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of loops</th>
<th>Binary search</th>
<th>Trie search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27169</td>
<td>1993</td>
</tr>
<tr>
<td>10</td>
<td>223974</td>
<td>2588</td>
</tr>
<tr>
<td>100</td>
<td>2188365</td>
<td>9224</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of loops</th>
<th>Binary search</th>
<th>Trie search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30110</td>
<td>2328</td>
</tr>
<tr>
<td>10</td>
<td>261135</td>
<td>3020</td>
</tr>
<tr>
<td>100</td>
<td>2550236</td>
<td>11431</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of loops</th>
<th>Binary search</th>
<th>Trie search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>36035</td>
<td>239</td>
</tr>
<tr>
<td>10</td>
<td>353907</td>
<td>955</td>
</tr>
<tr>
<td>100</td>
<td>3534087</td>
<td>8011</td>
</tr>
</tbody>
</table>
Table 8. Text Y (2000 words)

<table>
<thead>
<tr>
<th>Number of loops</th>
<th>Binary search</th>
<th>Trie search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3273</td>
<td>324</td>
</tr>
<tr>
<td>10</td>
<td>30857</td>
<td>1812</td>
</tr>
<tr>
<td>100</td>
<td>314067</td>
<td>16244</td>
</tr>
</tbody>
</table>

Table 9. Text Z (2000 words)

<table>
<thead>
<tr>
<th>Number of loops</th>
<th>Binary search</th>
<th>Trie search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42541</td>
<td>377</td>
</tr>
<tr>
<td>10</td>
<td>414514</td>
<td>2113</td>
</tr>
<tr>
<td>100</td>
<td>4147762</td>
<td>19061</td>
</tr>
</tbody>
</table>

It can be noticed that trie search depends on the linear way on the length of searched word and is very fast. In particular, this can be seen by comparing the “Text Y” and “Text Z” the longer search is connected with fact that the word niewybielalych is two characters longer than niewybielaly. We know that binary search depends logarithmically on dictionary size, but for particular text it depends on the position of searched word in the dictionary. Trie search generally for average text is 200 times faster than binary search.

5. Conclusions and future work

We examined two dictionary structures for text analysis in particular for indexing text. Tests have shown that the structure based on the trie makes searching many times faster. Both compared dictionary structures doubled all words in the dictionary in order to ensure the effective search. However, the structure using trie extremely utilizes RAM memory. The reason is that the alphabet has as many as 40 + 2 characters, and consequently every node of trie requires an array of such size. Many cells in of such arrays are empty. In the future work, we should investigate the structure that compromises the advantages of both tested structures.

REFERENCES