A Novel Information Search Approach for Languages Without Word Delimiters

Lianlong Wu

School of Electronic Engineering and Computer Science, Peking University, Beijing, China

Summary
In many languages there are no word delimiters among the text. It is very difficult to index articles in those languages. For example, Chinese information search engines always encounter a difficulty in segmentation of Chinese words from an article. In this paper, a suffix tree based searching approach is proposed to avoid the difficulty in segmentation of Chinese words. The suffix tree algorithms are studied and a set of optimal algorithms for index build are proposed. Based on the algorithms, a prototype of Chinese information search system is developed and applied to the Chinese Web Test collection with 100 GB Web pages (CWT-100g). The experimental results show that the system is capable of searching Chinese information without segmentation of Chinese words and the speed of index build is reduced to the theoretical limitation. part of summary.

Key words:
Search engine, segmentation of Chinese words, suffix tree, information system.

1. Introduction

An index mechanism is at the core of an information search system. Inverted list is a technique that is widely used as an index mechanism [1]. However, there are constraints on the use of such word-based lists in Chinese information systems. Since there are no delimiters to separate Chinese words, it is very difficult to segment words from a text. If one wants to apply inverted list, Chinese word segmentation has to be done at first. Firstly, it takes time to segment Chinese words. This affects the efficiency of index build. Secondly, the quality of word segmentation severely affects the quality of index.
In this paper, a full-text index mechanism based on suffix trees[2], [3] is applied to develop a new system, so the segmentation of Chinese words is not necessary. Based on properties of suffix trees, the search for phrases, sentences, and even more complicated searches, such as a whole paragraph search, are amazingly feasible. Moreover, accurate results can be achieved.

As index build is very time consuming, a set of optimal algorithms are studied in Section 3. The functionalities and interface of the new search system is described in Section 4. The experimental results are provided in Section 5. A conclusion is given in Section 6.

2. Problem of Inverted Files

Inverted files have been widely used as an index technique [1] in information search systems. Inverted files are a text index composed of a vocabulary and a list of occurrences. Inverted indices assume that the text can be seen as a sequence of words. In English, there are spaces to delimit words, but in Chinese text there is no space between words to separate them. Therefore, before an index is built, text information has to be segmented into Chinese words first.
The segmentation of Chinese words not only takes longer time than indexing itself, but also may make mistakes. For example, the Chinese phrase 'developing country' can be segmented in two sets of words shown in Figure 1. The segmentation at the right column is correct. The segmentation in the left column is wrong. The meaning is changed to 'to develop China'.

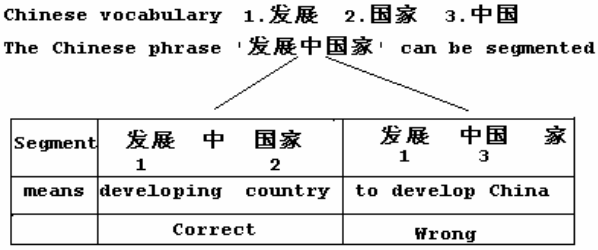

Figure 1. Difficulty of Chinese word segmentation.

3. Optimal Algorithms

3.1 Suffix Trees and Suffix Arrays

A suffix tree[2] is a data structure built over all the suffixes of a text. Suffix is a string that goes from a position to the end of the text. Each suffix is thus uniquely identified by the position. For example, *Suffix* (8) of"mississippi" is "ippi" as shown in Figure 2 (this example is from [4]).

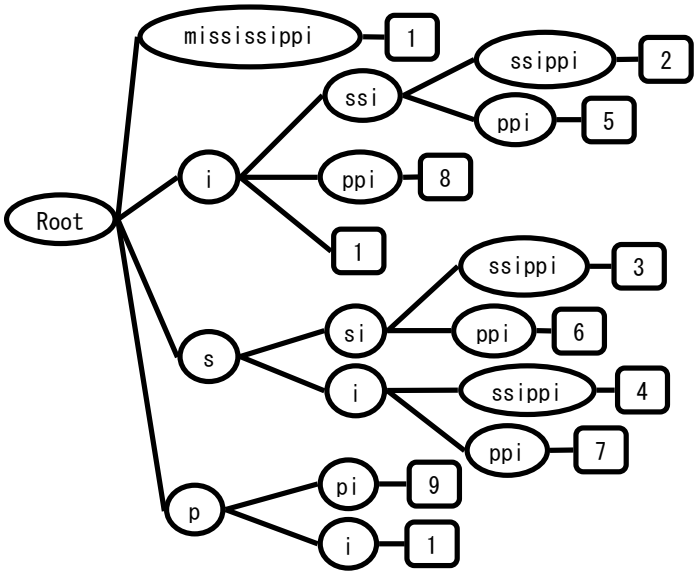

Figure 2. Suffix Tree of string "mississippi"

Suffixes in lexicographical order:

11: i
8: ippi
5: issippi
2: ississippi
1: mississippi
10: pi
9: ppi
7: sippi
4: sissippi
6: ssippi
3: ssissippi

Suffix Array	
Index	Value
1	11
2	8
3	5
4	2
5	1
6	10
7	9
8	7
9	4
10	6
11	3

Figure 3. Suffix Array of string "mississippi"

The data structure in a suffix tree can be easily applied to solve many problems on string operations such as those longest repeated substrings (2), (3), and (4). Any long substrings can easily be searched in the text. In order to take these advantages, a full-text index technique based on suffix trees is applied to develop the new search system. The system does not need to take time to do the segmentation of words. Therefore, a full-text index based on suffix trees can be built much faster than the index based on inverted files.

As suffix arrays provide the same function as suffix trees and occupy much less space, suffix arrays are proposed to implement the search system. A suffix array is simply an array containing all the pointers to the text suffixes listed in lexicographical order. For example, the suffix tree of string "mississippi" can be represented as a suffix array shown in Figure 3.

The Unicode is applied to represent Chinese characters. A content extraction module is developed to extract the contents from the Web pages. The text contents are stored in a Content File in the Unicode. The suffix array is stored as integers in the Index Array. The system produces files with file types as follows:

```
type  TIdx= Integer; // 32Bits Integer
      TCot = WideChar;          // Unicode Character
var
      CotFile : file of TCot; // Content File
      IdxFile : file of TIdx; // Index File
```

3.2 Suffix Array Building

Array $Rank_k$ is used for building the suffix array. $Rank_k[i]$ is the rank number of the $Suffix[i]$ according to the first k characters of each suffix. $Index_k$ is the suffix array corresponding to the $Rank_k$.

An optimal algorithm for building a suffix array is proposed as follows:

```
k ← 1;
repeat
    Quick Sort (Index_k); //according to first k characters of each suffix
   Count the Rank_2k Array;
    k ← 2k
until k≥n;
```

Where n is the length of the text, the time complexity for string comparisons in the Quick Sort is O (1), so the time complexity of Quick Sort is $O(n \cdot \log n)$. The variable k is increasing exponential; the time complexity of loop is log n. The total time complexity is $O(n \cdot \log^2 n)$.

For long query strings, it is not necessary to do a complete sort. The loop can be stopped when $k \geq m$, where m is the maximum length of query strings. This length is enough for the binary search. The time complexity is reduced to $O(n \cdot \log n \cdot \log m)$, where m is a small constant (m=16 or 32, in most cases), so the time complexity is $O(n \cdot \log n)$.

The Quick Sort can be replaced by Bucket Sort and Radix Sort. A optimal algorithm is proposed as follows:

k ← *1;*

Bucket Sort; //according to the first character only

Count Rank$_1$ Array;

repeat

Radix Sort (Index$_k$); //according to first k characters of each suffix

Count the Rank$_{2k}$ Array;

k ← *k*·*2;*

until k≥n;

Both the Bucket Sort and the Radix Sort are linear, so the total time complexity is reduced to O(*n*) that is theoretic lowest bound.

Additionally, an approach for rolling array is applied to recycle array memory. Scanning a static array is applied to replace dynamic queue so that the spaces for pointers are saved. Therefore, a high stable system is obtained. Because the Web pages are scanned only once to perform the extraction, the time complexity is O (n).

4. Development of Search System

A prototype of search system is developed in three separate modules and runs on single PC. First module is for extraction of content from Web pages. The *CotFile* is created. Second module is for index build. The optimal algorithms are applied in this module to build a suffix tree based index. A efficient search algorithm is applied to third module to search the information on the suffix tree based index. The index building interface is shown in Figure 4. The content files created by first module are listed in the left box. Select the content files and click bottom 'Suffix'. A suffix tree can be generated. Click bottom 'Index'. An index based on suffix tree is built. Click bottom 'Auto'. An index is automatically built for the selected content. Based on this interface, the information of the index can be shown. For example, a list of positions of URL is shown in the right box.

Figure 5. Shows searching interface of the prototype system. Multiple indexes are listed in the left box. The key words, phrase, or sentence can be input in the box beside the bottom 'search'. Results are shown in the middle boxes. The right box are listed the test collection provided in The CWT-100g[5].

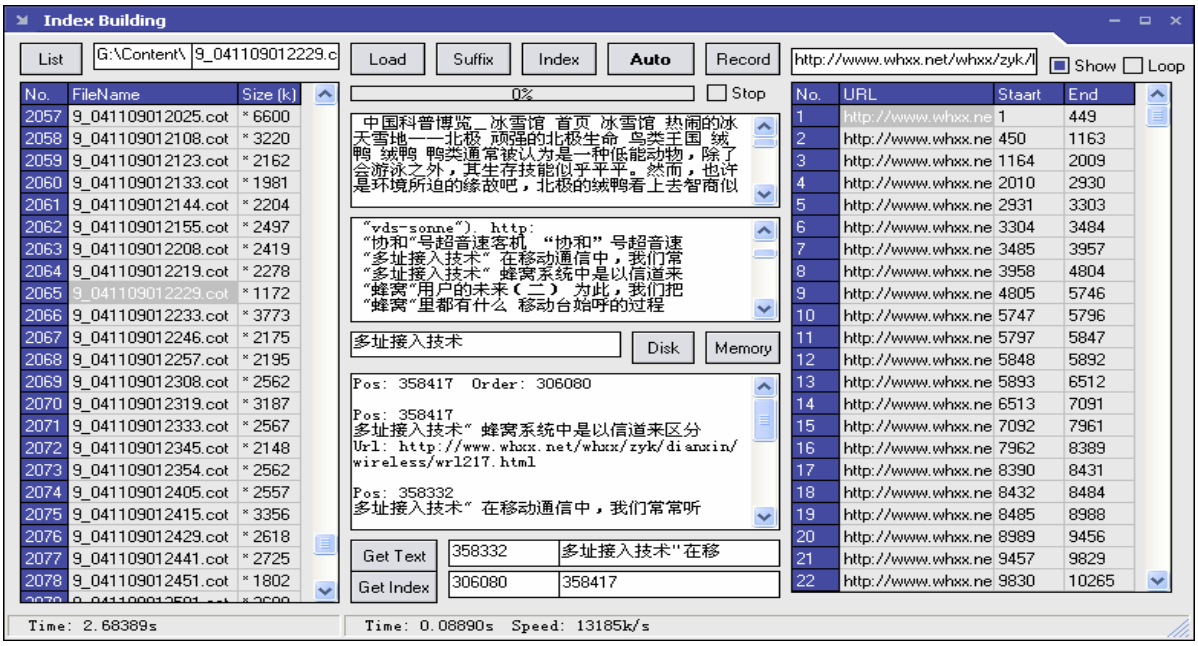

Figure 4. Interface of index build in the prototype system

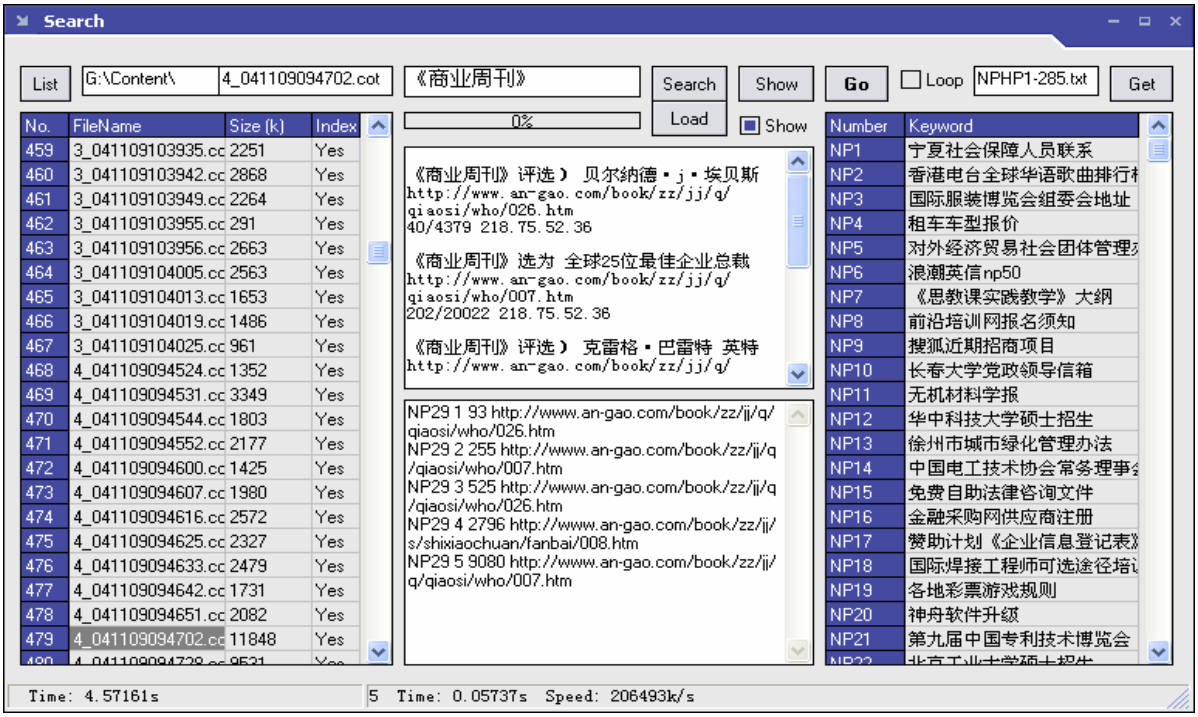

Figure 5. Searching interface in the prototype system

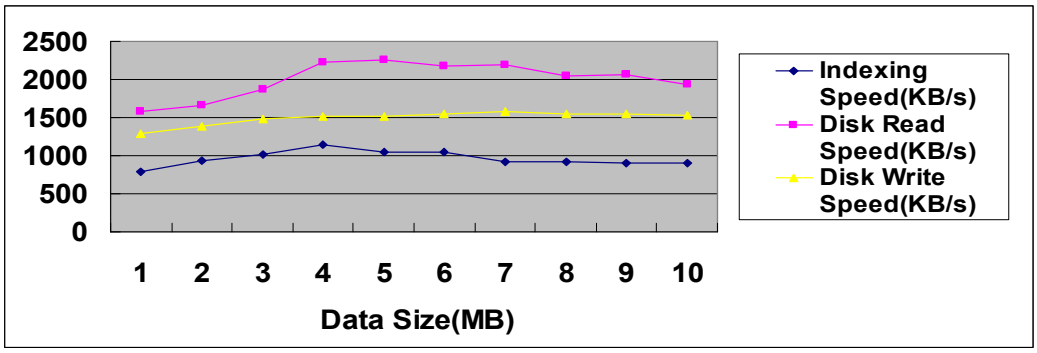

Figure 6. Indexing speed vs. hard disk speed

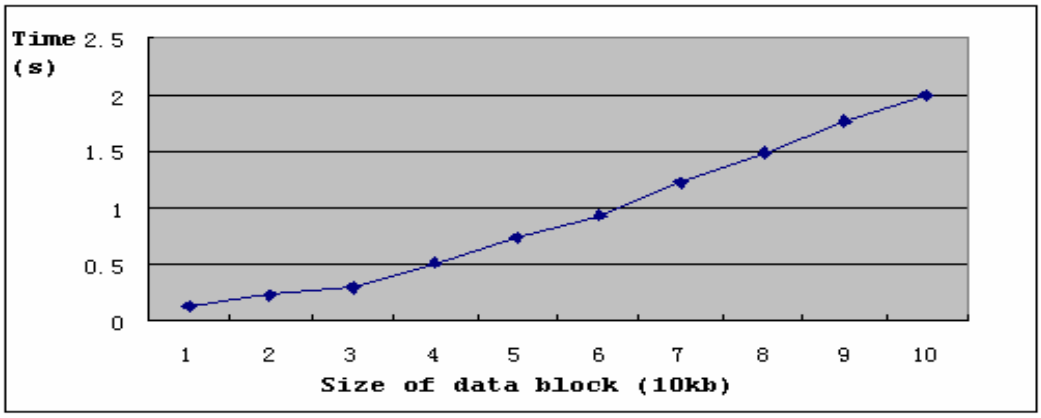

Figure 7. Indexing time shows linear increase with data size

5. Experimental Results

The new system was applied to the benchmark data set -- Chinese Web Test collection with 100 GB Web pages (CWT-100g) proposed by Network Group, Peking University [5]. The CWT-100g is a collection like TREC. It is composed of three parts: the documents, the queries, and relevance judgments. The CWT100g is designated as the test collection of SEWM-2004 Chinese Web Track.

Based on the benchmark data set, 285 items of home/named page finding tasks were done. Lists of URL were obtained corresponding to different tasks respectively. The total output for the 285 items of tasks is 4395 lines of URL. The running speed of the system was tested. The content was extracted from the 100GB Web pages with a speed of 3MB/s. Index build worked at a speed of 1MB/s. This is better than the speed of indexing based on inverted lists. As the fastest word segmentation speed is only 0.6MB/s, the indexing speed for inverted list based system is less than 0.6MB/s. The speed of indexing is highly dependent on hard disk read/write speed. The test results are shown in Figure 6. It can be seen that indexing speed is very close to hard disk speed. If a high-speed hard disk is applied, the higher indexing speed will be obtained. The relationship between indexing time and data size is shown in Figure 7. It can be seen that the indexing time shows linear increase with data size. For some other index mechanism, indexing time usually increases exponentially with data size. Based on this relationship, the indexing time for a huge number of Web pages can be predicted easily.

6. Conclusion

In this paper, a suffix tree was applied to index Chinese information in the information search system. As a result, the difficulties of segmentation of Chinese words can be avoided, and the system is independent of vocabulary library. The information can be searched not only by key words but also by any long substrings. The system supports searches with phrases, sentences, and even a paragraph. A set of improved algorithms was applied to the system so that the system can build index and search information fast. The index module was developed based on Unicode and is applicable to information in various languages, for example, Korea and Japanese languages. Based on this prototype system, a lot of further studies can be done. For example, if gene data set is input to the prototype system, frequency of gene sequence can be calculated by using the system.

Acknowledgments

I would like to thank the Computer Networks and Distributed Systems Laboratory in Peking University for providing the Chinese Web Test collection with 100 GB Web pages (CWT-100g).

References

[1] R. Baeza-Yates and B. Ribeiro-Neto, “Modern Information Retrieval”, Essex: ACM Press, 1999.
[2] E. M. McCreight, “A Space-economical Suffix Tree Construction Algorithm”, Journal of the ACM, 23(2): pp. 262--272, 1976.
[3] Ukkonen E, “On-line Construction of Suffix Trees”, Algorithmica, 14(3): pp. 249-260, 1995.
[4] http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Tree/Suffix/
[5] http://www.cwirf.org/en_index.html

Lianlong Wu is a student in the School of Electronic Engineering and Computer Science, Peking University, China. He is a student member of ACM. His research interesting includes information retrieval, P2P network, dynamic programming and network flow algorithm.