KEYWORD-DRIVEN SUFFIX ARRAYS FOR ON-LINE KEYWORD SEARCHING FROM DOCUMENTS IN CHINESE

Yanhua Zhang

School of Software Engineering of University of Science and Technology of China
Suzhou, China
cmlzyh@gmail.com

ABSTRACTS

On-line keyword searching from documents in Chinese tends to use inverted indexing as the main technique, which has its difficulties. Suffix Array is widely used for processing text in Western languages. However, it fails to get widely used in Chinese processing because of the speciality of Chinese. Suffix Array is a powerful tool. However it costs too much space. That is the major bottleneck of suffix Array. A data structure called Keyword-driven Suffix Array is proposed in this paper for on-line keyword searching from documents in Chinese, based on observation of on-line search pattern and traits of Chinese. Space efficiency is improved a lot using this data structure. When the document database is large enough, space efficiency is improved by about 5/6 using this data structure without sacrificing its time efficiency.

KEYWORDS

Chinese word segmentation, inverted index, Suffix Array, information retrieval, time/space efficiency

1. INTRODUCTION

1.1. Inverted-indexing mechanism

Given a document database consisting of a list of documents and also given a keyword, keyword searching is to find all occurrences of this word in the documents. Using an inverted file to search is now the mainly way for keyword searching from documents in Chinese. As stated in **Error! Reference source not found.**, an inverted file contains, for each term that appears anywhere in the database, a list of the numbers of the documents containing that term. To build an inverted file for documents in Chinese, corresponding techniques for word segmentation should be used because there is often no delimiter between two characters in Chinese text.

However, this method fails to work efficiently when the text is in Chinese. Firstly, the efficiency of searching depends on Chinese word segmentation **Error! Reference source not found.**. However, Chinese word segmentation has its own difficulties. For example, ambiguous segmentation is often encountered. Besides, this method doesn't work well for unstructured keywords.

1.2. Suffix Array and its bottleneck

Suffix Array is a method for on-line string searching proposed by Udi Manber and Gene Myers **Error! Reference source not found.**. It is a full-text indexed data structure and it indexes all suffixes. Thus suffix array can work independently of word segmentation.

Given a keyword K, on-line searching of K from a string S using suffix arrays can break into two phases. One is construction of the suffix array and the other is search.

Algorithms for linear-time construction of suffix array have been proposed in **Error! Reference source not found.-Error! Reference source not found.**.

The second phase is search. The longest common prefixes (lcps) of adjacent elements in the suffix array could be computed in linear time **Error! Reference source not found.** and the range minimum query problem (RMQ) could be solved in constant time **Error! Reference source not found.-Error! Reference source not found.**. When they are coupled, string searching can be answered in O(P + log N) time, where P is the length of K and N is the length of S.

Both of the two phases have got asymptotically optimal in theory. The fly in the ointment is that suffix arrays built for large text will cost too much space.

Compressed suffix arrays were proposed subsequently in **Error! Reference source not found.-Error! Reference source not found.**. Space efficiency is indeed improved using compressed suffix arrays. However, most such data structure manages to benefit from data compression. That means, some data compression techniques are firstly used to store suffix arrays and later corresponding decompression techniques are used to retrieve information. Thus space is obtained at the cost of time. Besides, it is often more complicated to implement and it is not very efficient for large alphabet.

A data structure called Keyword-driven Suffix Array is proposed in this paper for on-line keyword searching from documents in Chinese. Also the construction algorithm is given.

Differently from most compressed suffix arrays, this structure aims to decrease useless information in suffix arrays to improve space efficiency without sacrificing its time efficiency. What is more, all difficulties with inverted indexing method are not encountered using this new data structure. This data structure could also be used for languages which have the same traits with Chinese.

1.3. Overview

This paper is organized as follows. Section 3 gives a brief introduction to two basic structures - Suffix Array and Splay Tree for preparation of section 4. Section 4 explains the algorithm to build the new data structure called Keyword-driven Suffix Array. Section 5 gives the conclusion.

2. NOTATION

(1) Arrays index from 0 in the following sections.
(2) Suppose that S denotes a string with length n, both x and y are integers, $0 \leqslant x, y \leqslant n-1$, we will denote substring S[x]S[x+1]…S[y-1]S[y] as S[x..y].
(3) Given two strings, we use symbol " ○" to represent the concatenation of them. That is, xy ○ zw = xyzw, where "x", "y", "z" and "w" are all letters.
(4) We use "log" to denote binary logarithm.
(5) We use symbol "O" to denote the asymptotically upper bound.

3. BRIEF INTRODUCTION TO TWO DATA STRUCTURES

3.1. Introduction to Suffix Array

3.1.1. Definition of Suffix Array

Given a string S with length n, then its suffix array SA[0..n-1] is an array that satisfies the following inequality:

$$suffix(SA[i]) \leq suffix(SA[j]), \forall 0 \leq i < j \leq n-1 \qquad (1)$$

Suffix(i) denotes the suffix of S starting from position i, that is, suffix(i) = S[i..n-1].

3.1.2. Linear time suffix array construction

We will use DC-3 algorithm, which is proposed in 2006 **Error! Reference source not found.** for the construction of suffix arrays.

3.1.3. Search

Given two strings - S_1 and S_2, we use lcp(S_1, S_2) to represent the length of the longest common prefix of S_1 and S_2. LCP array is such an array that satisfies the following condition:

$$LCP[i] = lcp(suffix(SA[i]), suffix(SA[i-1])), \forall 0 < i \leq n-1 \qquad (2)$$

Besides, LCP[0] is defined to be 0.
LCP array can be constructed in O(n) time. Using LCP array and Range Minimum Query problem (RMQ), searching of a keyword with length p can be finished in O(p + log n) time.

3.2. Introduction to Splay Tree

Splay Tree is a self-adjusting form of binary search tree, which is developed by Daniel Dominic Sleator and Robert Endre Tarjan **Error! Reference source not found.**. On an n-node splay tree, all the standard search tree operations have an amortized time bound of O(log n) per operation.

4. Keyword-driven Suffix Arrays

4.1. Several observations

4.1.1. On-line search pattern

There exist many fields in the world, such as politics, economics, military affairs etc. Number of users varies a lot from one field to another. That is, some fields are more popular than others and the number of users interested in that field is larger than that in other fields. Besides, each user tends to take interest in one or more limited fields. On the other hand, each keyword is often closely associated with one certain field. Thus some keywords are accessed more frequently than others during a relatively long time interval.

4.1.2. Traits of Chinese

Most keywords searched by users are technical words or terms used frequently daily, though their lengths may vary. To study the ability of each Chinese character to start a keyword, firstly professional term lists in Chinese for computer science, economics and medicine are downloaded. Then a few Chinese characters are chose randomly and finally numbers of occurrences of all words (or terms) starting with these characters in each list are computed respectively. The result shows that the number of technical words that begin with a certain Chinese character is often not quite large and that technical words beginning with a certain Chinese character tend to be closely associated with several limited fields.

4.2. Description of the algorithm for constructing Keyword-driven Suffix Arrays for on-line keyword searching from documents in Chinese

Based on the observations shown above, algorithm for constructing Keyword-driven Suffix Array is stated as below. The algorithm could be broken into two phases – pre-processing and search.

4.2.1. Pre-processing

0) Construct a dictionary, Dict for Chinese characters frequently used. That is, map each Chinese character to a positive integer code. We use MAXChineseChar to denote the number of Chinese characters frequently used.

1) Get source string S.

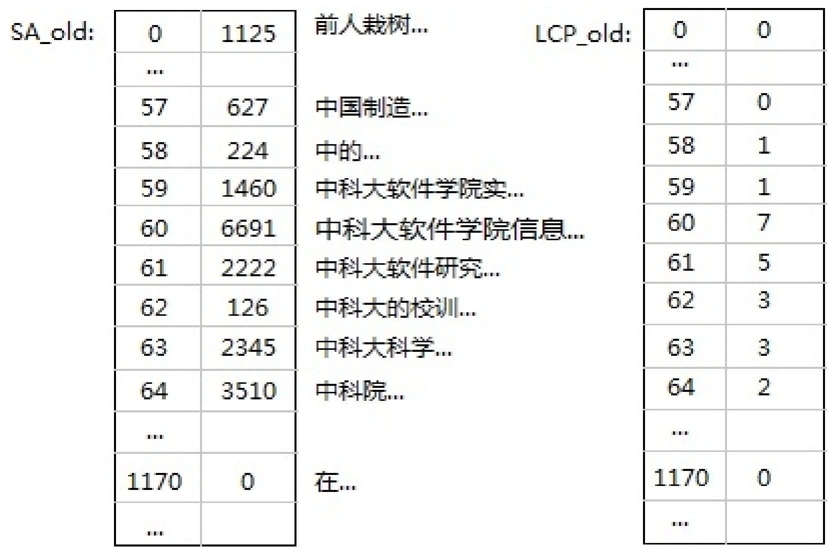
Concatenate all the documents to get the source string S, from which a keyword K is to be searched. Suppose that there exist s documents in the database D and we denote them as D1, D2, …, Ds respectively. Suppose sting S is of size N. Then we have equation (3).

$$S[0..N-1] = D = D1 \circ D2 \circ D3 \circ \ldots \circ DS \tag{3}$$

2) Construct the suffix array, *SA_old* for string *S*, using DC-3 algorithm. Next, construct corresponding LCP array, *LCP_old*. Samples of the two arrays are shown in figure 1.

3) Construct two arrays - *sub_SA* and *sub_LCP* both indexed from 1 to *MAXChineseChar*.

For i (1≤i≤*MAXChineseChar*), both *sub_SA*[i] and sub_*LCP*[i] are also arrays. *sub_SA*[i] stores the positions of all suffixes which begin with the corresponding character of code i sorted by the order of *Dict* code. Since *sub_SA*[i] consists of several consecutive items of array *SA_old*, its size can be computed using binary search. Correspondingly, array sub_*LCP*[i] is of the same length.



SA_old:

0	1125	前人栽树...
...		
57	627	中国制造...
58	224	中的...
59	1460	中科大软件学院实...
60	6691	中科大软件学院信息...
61	2222	中科大软件研究...
62	126	中科大的校训...
63	2345	中科大科学...
64	3510	中科院...
...		
1170	0	在...
...		

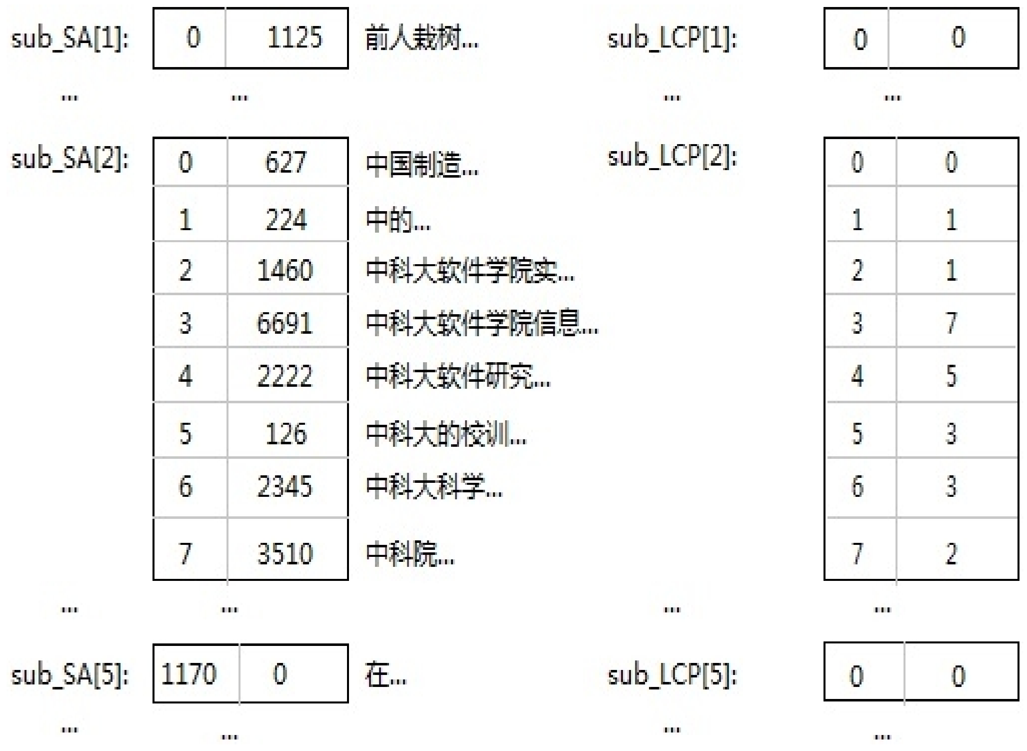
LCP_old:

0	0
...	
57	0
58	1
59	1
60	7
61	5
62	3
63	3
64	2
...	
1170	0
...	

Figure 1. Sample of suffix array and LCP array for source string S

4) Initialize both *sub_SA* and *sub_LCP*.

Scan array *SA_old* one by one from the beginning to the end. For i (0≤i≤*N*-1), which begins with 0, the character at position *SA_old*[i] in string S is *S*[*SA_old*[i]] and its *Dict* code is *Dict*[*S*[*SA_old*[i]]]. Thus element *SA_old[i]* can be set to the proper position of *sub_SA*[*Dict*[*S*[*SA_old*[i]]]]. That is, the element at the first position of array *sub_SA*[*Dic*t[*S*[*SA_old*[i]]]] (or array *sub_LCP*[*Dict*[*S*[*SA_old*[i]]]]) which hasn't been initialized could be initialized to be *SA_old*[i] (or *LCP_old*[i]). Repeat this procedure for all i, where 0≤i≤*N*-1. The two arrays are like figure 2 shows.



sub_SA[1]:	0	1125	前人栽树...	sub_LCP[1]:	0	0
...	...			...	...	
sub_SA[2]:	0	627	中国制造...	sub_LCP[2]:	0	0
	1	224	中的...		1	1
	2	1460	中科大软件学院实...		2	1
	3	6691	中科大软件学院信息...		3	7
	4	2222	中科大软件研究...		4	5
	5	126	中科大的校训...		5	3
	6	2345	中科大科学...		6	3
	7	3510	中科院...		7	2
...	...			...	...	
sub_SA[5]:	1170	0	在...	sub_LCP[5]:	0	0
...	...			...	...	

Figure 2. Sample of array *sub_SA* and array *sub_LCP*

5) Destruct both arrays of *SA_old* and *LCP_old.*

6) Construct an array of splay trees – *splay*[1.. *MAXChineseChar*].

Element *splay*[i] (1≤i≤MAXChineseChar) is a splay tree corresponding to character *C1*, whose *Dict* code is i. Each node in *splay*[i] corresponds to a character *C2* such that *C1*○*C2*○ xyz ("xyz" here represents some string. It may be empty as well) has been searched as a keyword. Each node in *splay*[i] has two fields – one is c*haracter* field, which is *C2* here and the other is a pair field of *(start, end)*, where *start* and *end* represent respectively the beginning and ending position of such an interval, *Intvl* of *sub_SA* that the suffix indexed at each position in *Intvl* begins with *C1*○*C2*. The c*haracter* field could be viewed as key for comparison between nodes. Initialize each splay tree *splay*[i] to be empty.

4.2.2. Search

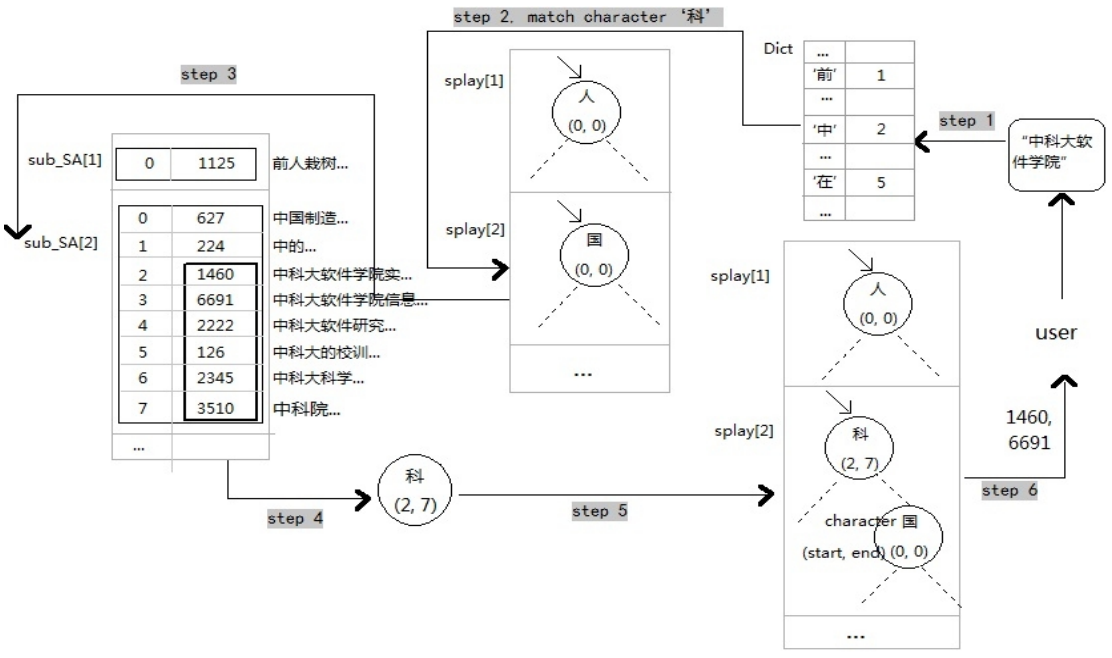


Figure 3. Sample of searching of keyword

The search phase is illustrated below. It works as figure 3 shows.

Suppose that the keyword, *K* being searched consists of m characters and we denote it as *K*[0..m-1].

To search *K*, first locate the splay tree *splay*[*Dict*[*K*[0]]], which is corresponding to the character *K*[0]. Then in this tree search a node whose c*haracter* field equals to *K*[1] using SEARCH operation for Splay Tree.

a) If such a node exists, suppose that the value of *(start, end)* field is (s, e). Then following suffix array based search algorithm as section 3.1.3 states, the remaining substring of *K* (*K*[2..m-1]) could be searched using *sub_SA*[*Dict*[*K*[0]]][s..e] and *sub_LCP*[*Dict*[*K*[0]]][s..e].

b) If such a node doesn't exist, substring *K*[0] ○*K*[1] can be searched using *sub_SA*[*Dict*[*K*[0]]] and *sub_LCP*[*Dict*[*K*[0]]]. Thus the starting and ending positions s and e can be obtained. Next, construct a new node P with *K*[1] as its c*haracter* field and with (s, e) as its field of *(start, end)*. Finally, node P can be inserted into *splay*[*Dict*[*K*[0]]] using INSERT operation for Splay Tree. Then search the remaining part of *K* as case a) illustrates.

With many keyword searched, each splay tree will expand quickly. When splay trees get large enough, each splay tree will get stable. It means keywords searched later will probably fall into its corresponding splay tree.

By now, the (*start*, *end*) field of each node of all splay trees can be replaced by a pair field which we name as (*SA*, *LCP*), where values of *SA* and *LCP* are *sub_SA*[*start*..*end*] and *sub_LCP*[*start*..*end*] respectively. By now, array *sub_SA*[1..*MAXChineseChar*] and *sub_LCP*[1..*MAXChineseChar*] are of no use any more and both of them can be destructed. Figure 4 gives a sample for splay trees in stable state.

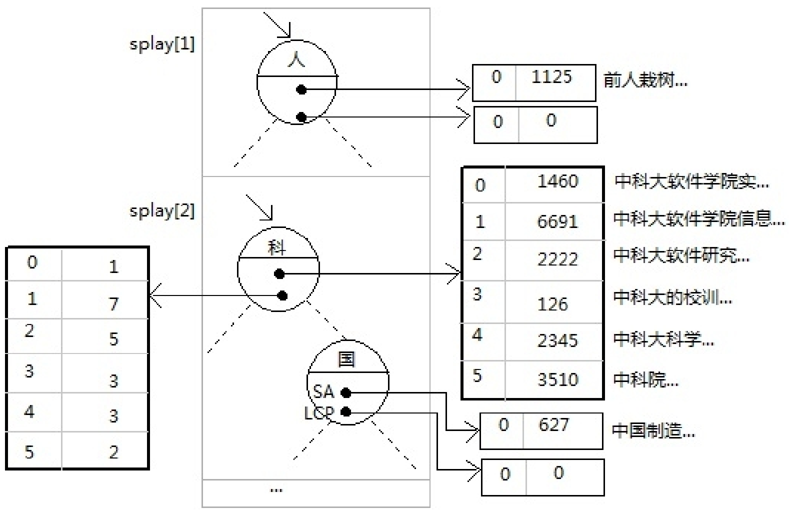


Figure 4. Sample of splay trees in stable state

This paper calls suffix arrays formed in this way *Keyword-driven Suffix Arrays.*
Then how to search a keyword, *K* after splay trees get stable?

To search *K*, first locate the splay tree *splay*[*Dict*[*K*[0]]], which is corresponding to the character *K*[0]. Then in this tree search a node whose c*haracter* field equals to *K*[1] using SEARCH operation for Splay Tree.

a) When such a node is found, name it as Q. The remaining substring of *K* (*K*[2..m-1]) can be searched using value of (*SA*, *LCP*) field of node Q.
b) Otherwise, construct a new node P with *K*[1] as its c*haracter* field. Then find all starting positions of occurrences of substring *K*[0] ○*K*[1] in string *S* using KMP algorithm **Error! Reference source not found.**. Next sort all suffixes indexed at the positions which are just obtained using multi-key quick sort method. Thus subfield *SA* of node P could be obtained and then subfield *LCP* could be computed by comparing adjacent items of *SA*. Using the two fields, all occurrences of *K* can be obtained. To maintain the corresponding splay tree *splay*[*Dict*[*K*[0]]], node P should be inserted into it first. Then use a rotation operation to move node P to be father of the node F, who is father of P before. Then delete node F and use SPLAY operation to move node P to the root.

Notes: case a will happen at a percentage much higher than case b.
By experiment when the size of each splay tree reaches 167 on average (height of each splay tree is 8 on average), splay trees get stable. The hit rate will then reach 95%, which could be viewed as a threshold value.

Suppose that there are 5,000 Chinese characters frequently used. According to statistics, at most one in six suffixes are those that frequently searched keywords begin with. Then length of both *SA* and *LCP* for each node in all splay trees is at most

$$N\times(1/6)\times(1/5000)\times(1/167)\approx N/5000000 \quad (4)$$

4.3. Performance analysis

4.3.1. Time efficiency

i. Pre-processing
Time used in this phase is O(N).

ii. Search

a) Before splay trees become stable
Time used for searching is O(m+log(N/5000)).

b) After splay trees get stable
Time efficiency of search in this phase is about O(m + log (N/5000000)).
To make analysis below more concise, we will use the number of CPU operations. The "m" item and the multiplication factors within log could be neglected when N is large enough. The number of CPU operations for maintaining splay trees can be viewed as 8 since each splay tree is of height of 8 on average.

Thus when N is large enough, for 95 percent keyword searching requests, the number of CPU operations is log (N/5000000) + 8 using this method. That is less than log N, which is the number of CPU operations using Suffix Array.

4.3.2. Space efficiency

a) Pre-processing
Space used is O(N), the same as that used using Suffix Array.

b) Search

Extra space occupied by pointers of nodes of splay trees could be neglected when N is large enough.

1) Before splay trees get stable, Space used is O(N), the same as that used using suffix arrays.
2) After splay trees get stable, space occupied by (SA, LCP) field of all nodes in all splay trees is about O((1/6)*N).

Space used by this data structure is decreased by about 5/6 after splay trees get stable.

5. CONCLUSION

Space efficiency is improved using this new data structure because it is more strongly structured than Suffix Array.

ACKNOWLEDGEMENTS

Firstly, I would like to show my deepest gratitude to Mr. Chenxi Shao and Mr. Liusheng Huang for their professional guidance. Secondly, I appreciate Wei Zhou for his help with my English during these years. Finally, I thank my father, though he has been gone for 9 years and my mother.

REFERENCES

[1] Alistair Moffat & Justin Zobel, (1996) "Self-indexing inverted files for fast text retrieval", ACM Transactions on Information Systems, Vol. 14, No. 4, pp. 349-379.

[2] Richard Sproat & Thomas Emerson, (2003) "The First International Chinese Word Segmentation Bakeoff", In Proceedings of the Second SIGHAN Workshop on Chinese Language Processing, pp. 133-143.

[3] Udi Manber & Gene Myers, (1993) "Suffix arrays: A new method for on-line string searches", SIAM Journal on Computing, Vol. 22, No. 5, pp. 935-948.

[4] Dong Kyue Kim, Jeong Seop Sim, Heejin Park & Kunsoo Park, (2003) "Linear-Time Construction of Suffix Arrays", in Proceedings of the 14th annual conference on Combinatorial pattern matching, pp.186-199.

[5] Pang Ko & Srinivas Aluru, (2003) "Space Efficient Linear Time Construction of Suffix Arrays", in Proceedings of the 14th annual conference on Combinatorial pattern matching, pp. 200-210.

[6] Juha KÄRKKÄINEN, Peter Sanders & Stefan Burkhardt, (2006) "Linear work suffix array construction", Journal of the ACM (JACM) , Vol.53, No. 6, pp. 918-936.

[7] Ge Nong, Sen Zhang & Wai Hong Chan, (2009) "Linear Suffix Array Construction by Almost Pure Induced-Sortings", in Data Compression Conference, pp. 193-202.

[8] KASAI, T., LEE, G., ARIMURA, H., ARIKAWA, S. & PARK, K., (2006) "Linear-time longest-common prefix computation in suffix arrays and its applications", in Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching, pp. 181-192.

[9] Bender, Michael A, Farach-Colton & Martin, (2000) "The LCA problem revisited", in Proceedings of the 4th Latin American Symposium on Theoretical Informatics, pp. 88-94.

[10] Johannes Fischer & Volker Heun, (2006) "Theoretical and Practical Improvements on the RMQ-Problem, with Applications to LCA and LCE", in Proceedings of the 17th Annual conference on Combinatorial Pattern Matching, pp. 36-48.

[11] Demaine, Erik, Gad Landau & Oren Weimann, (2009) "on Cartesian Trees and Range Minimum Queries", in Proceedings of the 36th International Colloquium on Automata, Languages and Programming, pp. 341-353.

[12] Roberto Grossi & Jeffrey Scott Vitter, (2005) "Compressed suffix arrays and suffix trees with applications to text indexing and string matching", SIAM Journal on Computing, Vol. 35, No. 2, pp. 378-407.

[13] Roberto Grossi, (2011) "A quick tour on suffix arrays and compressed suffix arrays", Theoretical Computer Science, Vol. 412, No. 27, pp. 2964-2973.

[14] Daniel Dominic Sleator & Robert Endre Tarjan, (1985) "Self-adjusting binary search trees", Journal of the ACM (JACM), Vol. 32, No.3, pp. 652-686.

[15] Donald Knuth, James H.Morris & Vaughan Pratt, (1977) "Fast Pattern in strings", SIAM Journal on Computing, Vol. 6, No. 2, pp. 323-350.

Authors

I am now a student of School of Software Engineering of University of Science and Technology of China. I graduated from Hebei Polytechnic University in July 2004, getting my bachelor's degree majoring in computer science and technology. From 2004 to 2007 I worked within Hebei Datang Information and Technology Company Ltd. During these years I began to take interest in algorithm designing and analysis and I read several famous books such as Introduction to Algorithms to learn algorithm comprehensively and systematically. To get more professional guidance, I entered School of Software Engineering of University of Science and Technology of China in July 2010. I began to know suffix array a bout 1 year ago. I read several papers about it and realized that it could be improved with a little change to search on-line from Chinese documents. Thus I write this paper.