International Journal of Database Theory and Application
Vol.10, No.1 (2017), pp.207-212
http://dx.doi.org/10.14257/ijdta.2017.10.1.19

A Numerical Method for Suffix Array Index Compression

Baomin Xu[1], Jie Huang[2] and Yang Yang[1]

[1]*School of Computer and Information Technology, Beijing Jiaotong University Beijing 100044*
bmxu@bjtu.edu.cn, 13281167@bjtu.edu.cn
[2]*College of National Secrecy, Harbin Engineering University, Harbin 150001*
2585672555@qq.com

Abstract

Suffix arrays is versatile data structures playing a key role in numerous string processing applications such as the data structure can be used to represent the given DNA strings. However, the most serious drawback of suffix arrays is their size, namely space usage. In this paper, we propose a new suffix array compression technique, i.e., numerical method for suffix array index compression, for the problem. With the method, we will translate DNA bases characters ATGC to the corresponding integer number 1234. The experimental results show that the numerical method for suffix array index compression not only can greatly compress the memory space of suffix array, but also can retain the quick search characteristics of suffix array.

Keywords: *suffix tree; suffix array; DNA; index compression*

1. Introduction

Since the discovery of the molecular double-helix structure of DNA, the molecular biology has made tremendous progress [1]. With the conduct of various biological sequencing, the DNA sequences data is growing exponentially [2], and the researchers usually do various search in gene sequences and protein sequences. Therefore, the design of efficient index structure of biological sequences and researching the index compression technology has become an important field of bioinformatics.

Large-scale data is the typical feature of biological sequence database [3,4]. The sequence data of GenBank database continues to grow at an exponential rate growing, doubling every 14 months [5]. At present, GenBank database has about 100 billion bases from 70,000 species, the daily querying more than 40,000 times [6], which requires a very efficient query method. Indexing the biological sequence is a way to improve efficiency approach.

There are some method can be used to index biological sequence, such as suffix tree [7-8], suffix array [9], Q-gram [10], Q-samples [9-10], and so on. Different method has different time-complexity and space-complexity. Suffix tree is discussed in literatures [11-12], which using tree structure storing base string. The query in suffix tree is fast, and the time-complexity is proportional to the query string length. However, to store the suffix tree requires a lot of memory space, and the average size of each base character is about 8.5 byte [11], which is only suitable for small sequence. Suffix array is similar to suffix tree, which can achieve the similar function with suffix tree and has the similar time-complexity. Algorithmically, suffix array is interchangeable to suffix tree, which is better than the suffix tree in the store space. When constructing the suffix array, we should first take the entire suffix string in alphabetical order and then arrange the results into the array.

The space-consuming of suffix array is less than suffix tree's, but it still takes up a lot of memory. Therefore, this paper puts forward a numerical method for suffix array index

ISSN: 2005-4270 IJDTA

compression. Making use of the properties of OS and Java programming language, this method can greatly compress the memory space of suffix array, and retains the quick search characteristics of suffix array.

2. Preliminaries

2.1 Suffix Tree

To string *S* of length *m*, its suffix tree *T* is a rooted directed tree. The suffix tree has m leaves which labeling 1 to *m*. The suffix tree's node has at least two children except root and leaves, and each branch contains a non-empty substring of S. Form one node, there are no two nodes have the same branch labeling. The key feature of suffix tree is that the set of all between root node and every leaf node *i* is the suffix from location *i* of *S*, *i.e.* *S*[*i*…*m*] [8].

For any string *S*, it can establish its suffix tree *T* using linear-time. The classic algorithm to establish suffix tree has Ukkonen method and Weiner method. Weiner is the first man to establish using linear-time, whose algorithm has historical importance and different tectonic thinking. Ukkonen method is as fast as the Weiner method when establishing a suffix tree, which uses less space.

If a string S of length *m* and its suffix tree is *T*, for any pattern string *P* which length is *n*, you can find the pattern string *P* all appear location in *S* with $O(n+k)$ time (k is the total number of pattern string *P* appear times in string S).

Figure1 is a suffix tree of one string.

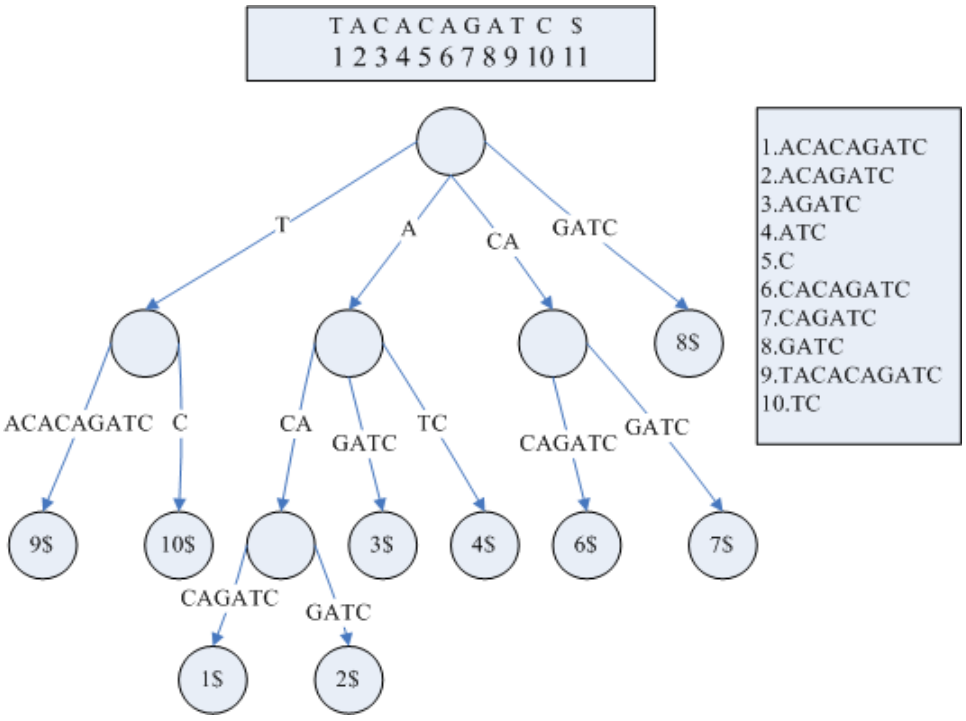

Figure 1. Suffix Tree on TACACAGATC$

2.2 Suffix Array

String *T* of *m* chars, its suffix array *pos* is an array which value is between 1 to *m*. Suffix array point to *m* suffix string of *T* in alphabetical order [8]. The suffix array only storage the Integer numbers not the suffix string of *T*. Therefore, the memory space of suffix array is small. For a string of length *m*, its suffix array just saves m integer number of length log*m* byte.

The worst time for directly establishing suffix array is $O(m\log m)$, and the average time is $O(n\log\log m)$ [14]. The author using suffix sort to establish the suffix array and this algorithm time-complexity is linear. When using suffix array to search, you can use binary search method and also add a $O(\log m)$ time factor.

The suffix array of sequence TACACAGATC is shown in Figure2.

i	pos[*i*]	$A_{pos[i]}$
1	2	ACACAGATC
2	4	ACAGATC
3	6	AGATC
4	8	ATC
5	10	C
6	3	CACAGATC
7	5	CAGATC
8	7	GATC
9	1	TACACAGATC
10	9	TC

Figure 2. Suffix Array

3. Numerical Method for Index Compression

Suffix arrays occupies less space than suffix tree, but there are still a lot of space. In Java programming language of 32-bit computer system, char data type use 2-byte space and int data type use 4-byte space. We can achieve the purpose of suffix array compression by translating DNA base character ATGC to 1234.

Numerical compression method is as follows:

1. Preprocessing: translate character ATGC to numerical character.

For example:

‘A’—>‘1’ , ‘T’—>‘2’ , ‘G’—>‘3’ , ‘C’—>‘4’

2. Translate numerical string to integer number

For example:

“A”—>1, “T”—>2, “G”—>3, ‘C’—>4, “ATGC” —>1234

3. Storage by using integer number type data

Sequence TACACAGATC can translate as shown in Figure3.

Seen from Figure3, we can see the total size of suffix array of sequence TACACAGATC usage of memory space is 108byte, and numerical suffix array is 44byte, and the compression rate is 59%. In Java Virtual Machine, the range of int data type is -2147483648 to 2147483647, and the highest bit is 2. So in numerical process, the maximum string length is 9, the largest combined number 444444 444 <2147483647.

i	pos[i]	(Char[]) $A_{pos[i]}$	memory space
1	2	ACACAGATC	18 byte
2	4	ACAGATC	14 byte
3	6	AGATC	10 byte
4	8	ATC	6 byte
5	10	C	2 byte
6	3	CACAGATC	16 byte
7	5	CAGATC	12 byte
8	7	GATC	8 byte
9	1	TACACAGATC	20 byte
10	9	TC	2 byte

i	pos[i]	(int) $A_{pos[i]}$	memory space
1	2	141413124	4 byte
2	4	1413124	4 byte
3	6	13124	4 byte
4	8	124	4 byte
5	10	4	4 byte
6	3	41413124	4 byte
7	5	413124	4 byte
8	7	3124	4 byte
9	1	24141312+4	8 byte
10	9	24	4 byte

Figure 3. Numerical Suffix Array

In numerical process, set s is translating string and its length is len, and compression rate p is shown in Figure4.

$$p=\begin{cases}\dfrac{2-4}{2}=-100\% & (\ while\ len=1\ )\\ \dfrac{2*len\text{-}4}{2*len}=\dfrac{len\text{-}2}{len}=1-\dfrac{2}{len} & (\ while\ 2=<len<=9\ )\\ \dfrac{2*len\text{-}4*(M+1)}{2*len}=\dfrac{len\text{-}2*(M+1)}{len}=1--\dfrac{2*(M+1)}{len} & (\ M=\dfrac{len}{9}\ while\ len>=10\ )\end{cases}$$

Figure 4. Compression Rate

4. Results and Discussion

The experiment sequence data is produced by random generator. Test data has 10 parts: 100,000bp,200,000bp,300,000bp,400,000bp,500,00bp,600,000bp,700,000bp,800,000bp,900,000bp,1,000,000bp. All the applications are programmed by Java, and the environment for run are as follows: OS: Windows XP, CPU: Intel Core Duo T2250 1.73GHz, RAM: 1.5G, DISK:80G.

From Figure5, we can see that numerical method is slower than suffix tree. The construction method of numerical method is same to the suffix tree expect the node value type. Suffix tree node value is ATGC of character type, and numerical method node value is 1234 of integer number type. When translate the string type to integer number type, there requires extra time.

From Figure6, we can see the usage of space size: the suffix tree is largest; the numerical suffix array is smallest. One base usage 2-byte space, from experiment we can get that: one base requires about 27-byte space on average of suffix tree; one base requires about 11-byte space on average of suffix array; one base requires about 6-byte space on average of numerical suffix array. The numerical method is most excellent.

5. Conclusion

Our numerical method not only maintains the quick search feature of suffix array, but also compress the suffix array, which reduce the requirement for memory space. The future work will mainly focus on the approximate matching query to find the longest common subsequence on the suffix array and numerical suffix array.

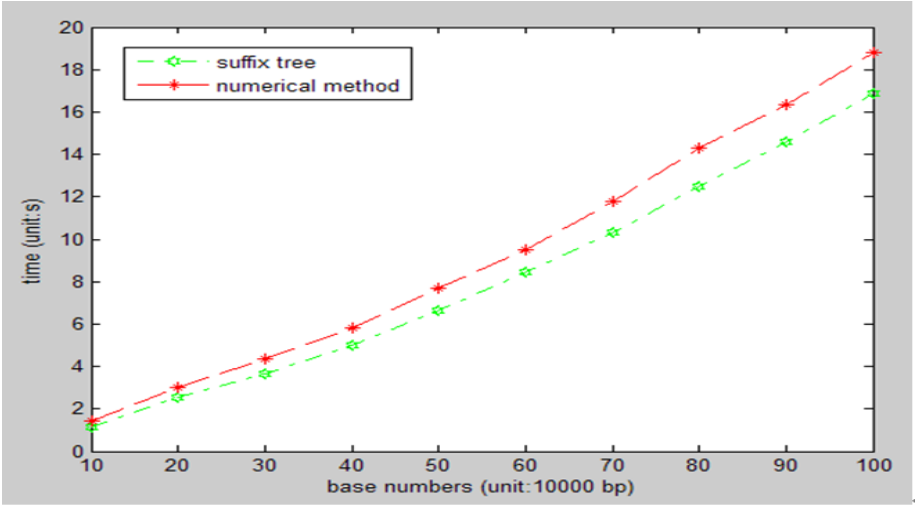

Figure 5. Time of Index Construction

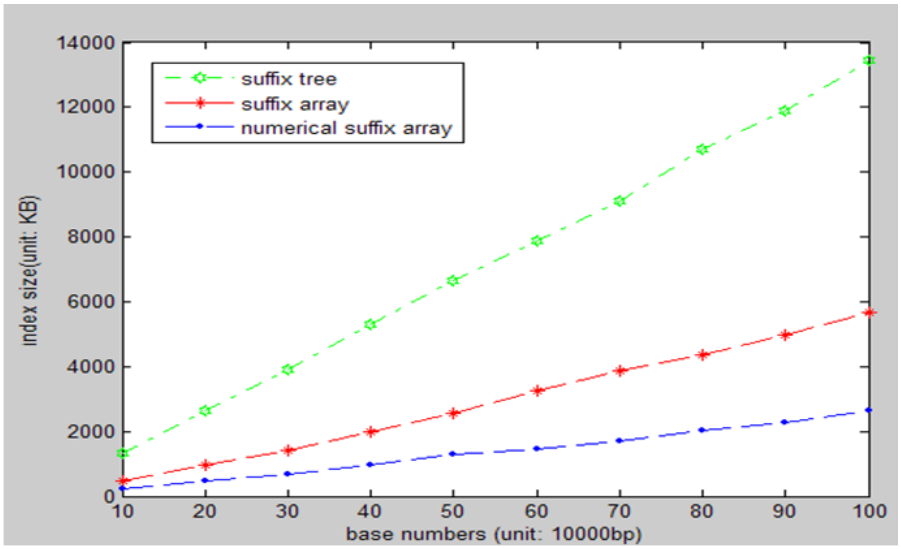

Figure 6. Size of Index

Acknowledgements

This work is supported by the Fundamental Research Funds for the Central Universities (2015JBM035).

References

[1]. Kiełbasa, Szymon M., Raymond Wan, Kengo Sato, Paul Horton, and Martin C. Frith. "Adaptive seeds tame genomic sequence comparison." Genome research 21, 3 (2011): 487-493.
[2]. Kreft, Sebastian, and Gonzalo Navarro. "On compressing and indexing repetitive sequences." Theoretical Computer Science 483 (2013): 115-133.
[3]. Shrestha, Anish Man Singh, Martin C. Frith, and Paul Horton. "A bioinformatician's guide to the forefront of suffix array construction algorithms." Briefings in bioinformatics 15, 2 (2014): 138-154.
[4]. Xu, B., J. Gao, and C. Li. "An efficient algorithm for DNA fragment assembly in MapReduce. " Biochemical & Biophysical Research Communications 426.3(2012):395-398.
[5]. NCBI Database. URL: http://www.ncbi. nlm.nih.gov/Database/index.html, 2010.
[6]. Y Larsson, N. Jesper, Kasper Fuglsang, and Kenneth Karlsson. "Efficient Representation for Online Suffix Tree Construction." arXiv preprint arXiv:1403.0457 (2014).
[7]. Crochemore, Maxime, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. "Order-Preserving Suffix Trees and

Their Algorithmic Applications." arXiv preprint arXiv:1303.6872 (2013).
[8]. Mansour, Essam, Amin Allam, Spiros Skiadopoulos, and Panos Kalnis. "ERA: efficient serial and parallel suffix tree construction for very long strings." Proceedings of the VLDB Endowment 5, 1 (2011): 49-60.
[9]. Burcsi, Péter, Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. "On approximate jumbled pattern matching in strings." Theory of Computing Systems 50, 1 (2012): 35-51.
[10]. Saabni, Raid, Alex Bronstein, and Israel Qarea. "Fast Keyword Searching Using'BoostMap'Based Embedding." In ICFHR, pp. 734-739. 2012.
[11]. Tian Y, Tata S, Hankins R A, Patel J M. Practical Methods for Constructing Suffix Trees. The VLDB Journal, 2005, 14(3): 281-299.
[12]. Barsky, Marina, Ulrike Stege, and Alex Thomo. "Suffix trees for inputs larger than main memory." Information Systems 36, 3 (2011): 644-654.
[13]. Gog, Simon, Alistair Moffat, J. Culpepper, Andrew Turpin, and Anthony Wirth. "Large-scale pattern search using reduced-space on-disk suffix arrays." (2013): 1-1.
[14]. Nong, Ge, Sen Zhang, and Wai Hong Chan. "Two efficient algorithms for linear time suffix array construction." Computers, IEEE Transactions on 60, 10 (2011): 1471-1484.

Authors

Baomin Xu, is an associate professor in School of Computer and Information Technology, Beijing Jiaotong University, China. His research interests include Large Scale Data Analysis Based-on Cloud Computing, and Distributed Computing. Until now he has published more than 50 research papers in referred journals and proceedings.

Jie Huang, is an undergraduate student at the College of National Secrecy, Harbin Engineering University. Her current research interests include information security and big data processing.

Yang Yang, is an undergraduate student at the school of computer and information technology, Beijing Jiaotong University. Her current research interests include cloud computing and big data processing.

