Vestnik of Don State Technical University. 2018. Vol. 18, no. 4, pp. 449–454. ISSN 1992-5980 eISSN 1992-6006
Вестник Донского государственного технического университета. 2018. Т. 18, № 4. С. 449–454. ISSN 1992-5980 eISSN 1992-6006

УДК 004.451.54 https://doi.org/10.23947/1992-5980-2018-18-4-449-454

Parallel construction of binary tree based on sorting[*]

Ya. E. Romm[1], D. A. Chabanyuk[2]**
[1, 2] Taganrog Chekhov Institute, Rostov State University of Economics (RINH) branch, Rostov-on-Don, Russian Federation

Параллельное построение двоичного дерева на основе сортировки[***]

Я. Е. Ромм[1], Д. А. Чабанюк[2]**
[1, 2] Таганрогский институт имени А. П. Чехова (филиал) Ростовского государственного экономического университета (РИНХ)", г. Таганрог, Российская Федерация

Introduction. Algorithms for the parallel binary tree construction are developed. The algorithms are based on sorting and described in a constructive form. For the N element set, the time complexity has $T(R)=O(1)$ and $T(R)=O(\log_2 N)$ estimates, where $R=(N^2-N)/2$ is the number of processors. The tree is built with the uniqueness property. The algorithms are invariant with respect to the input sequence type. The work objective is to develop and study ways of accelerating the process of organizing and transforming the tree-like data structures on the basis of the stable maximum parallel sorting algorithms for their application to the basic operations of information retrieval on databases.
Materials and Methods. A one-to-one relation between the input element set and the binary tree built for it is established using a stable address sorting. The sorting provides maximum concurrency, and, in an operator form, establishes a one-to-one mapping of input and output indices. On this basis, methods for the mutual transformation of the binary data structures are being developed.
Research Results. An efficient parallel algorithm for constructing a binary tree based on the address sorting with time complexity of $T(N^2)=O(\log_2 N)$ is obtained. From the well-known analogues, the algorithm differs in structure and logarithmic estimation of time complexity, which makes it possible to achieve the acceleration of $O(N^\alpha)$, $\alpha \ge 1$ order analogues. As an advanced version, an algorithm modification, which provides the maximum parallel construction of the binary tree based on a stable address sorting and a priori calculation of the stored subtree root indices is suggested. The algorithm differs in structure and estimation of $T(1)=O(1)$ time complexity. A similar estimate is achieved in a sequential version of the mod-

Введение. Разработаны алгоритмы параллельного построения двоичного дерева. Алгоритмы выполнены на основе сортировки и описаны в конструктивной форме. Для множества из N элементов временная сложность имеет оценки $T(R)=O(1)$ и $T(R)=O(\log_2 N)$, где число процессоров $R=(N^2-N)/2$. Дерево строится со свойством единственности. Алгоритмы инвариантны относительно вида входной последовательности. Целью работы являлась разработка и исследование способов ускорения процесса организации и преобразований древовидных структур данных на основе алгоритмов устойчивой максимально параллельной сортировки для их применения к базовым операциям информационного поиска в базах данных.
Материалы и методы. Взаимно однозначное соответствие множества входных элементов и построенного для него двоичного дерева устанавливается при помощи устойчивой адресной сортировки. Сортировка обладает максимальным параллелизмом, в операторной форме устанавливает взаимно однозначное соответствие входных и выходных индексов. На этой основе разрабатываются методы взаимного преобразования двоичных структур данных.
Результаты исследования. Получен эффективный параллельный алгоритм построения двоичного дерева на основе адресной сортировки с временной сложностью $T(N^2)=O(\log_2 N)$. От известных аналогов алгоритм отличается структурой и логарифмической оценкой временной сложности, позволяющей достигать ускорения аналогов порядка $O(N^\alpha)$, $\alpha \ge 1$. В качестве усовершенствованного варианта предложена модификация алгоритма, обеспечивающая максимально параллельное построение двоичного дерева на основе устойчивой адресной сортировки и априорного вычисления хранимых индексов корней поддеревьев. Алгоритм отличается структурой и оценкой временной сложности $T(1)=O(1)$. Аналогичная оценка достигается в последовательном варианте модифициро-

[*] Работа выполнена в рамках инициативной НИР.
[**] E-mail: romm@list.ru, denchabanyuk@gmail.com
[***] The research is done within the frame of independent R&D.

ified algorithm, which allows obtaining the acceleration of known analogs of $O(N^{\alpha})$ $\alpha > 1$ order.
Discussion and Conclusions. The results obtained are focused on the creation of effective methods for the dynamic database processing. The proposed methods and algorithms can form an algorithmic basis for an advanced deterministic search on the relational databases and information systems.

ванного алгоритма, что позволяет достигать ускорения известных аналогов порядка $O(N^{\alpha})$, $\alpha > 1$.
Обсуждения и заключение. Полученные результаты направлены на организацию эффективных способов динамической обработки баз данных. Предложенные способы и алгоритмы могут составить алгоритмическую основу для ускоренного детерминированного поиска в реляционных базах данных и информационных системах.

Keywords: data structures, data processing algorithms, binary tree, algorithms for parallel sorting.

Ключевые слова: структуры данных, алгоритмы обработки данных, двоичное дерево, алгоритмы параллельной сортировки.

For citation: Ya.E. Romm, D.A. Chabanyuk. Parallel construction of binary tree based on sorting. Vestnik of DSTU, 2018, vol. 18, no.4, pp. 449-454. https://doi.org/10.23947/1992-5980-2018-18-4-449-454

Образец для цитирования: Ромм, Я. Е. Параллельное построение двоичного дерева на основе сортировки / Я. Е. Ромм, Д. А. Чабанюк // Вестник Дон. гос. техн. ун-та. — 2018. — Т. 18, № 4. — С. 449-454. https://doi.org/10.23947/1992-5980-2018-18-4-449-454

Introduction. There is a tendency to the convergence of parallel information processing technologies and various processor architectures in the field of modern high-performance computing. Despite the variety of processor architectures and ways of presenting information, the idea of parallel processing is one of the most important tasks of computer science to increase the data-rate. To accelerate processing speed, the authors propose to use a stable address sorting algorithm with maximum parallelism.

Method of parallel construction of a binary tree. For $A = (a_0, a_1, \ldots, a_{n-1})$ array, the comparison matrix is developed according to [1, 2]. a_{ij} element of this matrix is defined as

$$a_{ij} = \operatorname{sign}(a_j - a_i) = \begin{cases} +, & a_j > a_i \\ 0, & a_j = a_i \\ -, & a_j < a_i \end{cases},$$

where $i, j = 1, 2, \ldots, n$.

a_j element in $C = (c_0, c_1, \ldots, c_{n-1})$ sorted array gets the number $k = \sum_{i=1}^{n} a_{ij}$, where $a_{ij} \geq 0$ at $i \leq j$, $a_{ij} > 0$ at $i > j$. All comparisons are mutually independent; the sorting is stable and as parallel as possible with the estimate of $T\left(\frac{N^2 - N}{2}\right) = O(1)$ time complexity. On this basis, you can perform a parallel construction of a binary tree [3, 4].

Suppose we are given a set of N elements, all elements of which are represented as a single-dimension array. On the set, $\leq$ ordering relation is assumed. It is required to convert the array into a binary tree. For this, the described array sorting is performed. C medial array cell has $j_{\text{ср}} = \left\lceil \frac{N}{2} \right\rceil$ index and is taken as the root of the tree [3]. All C array components to the left of $C_{j_{\text{ср}}}$ form a left subtree (left subarray). The components to the right of $C_{j_{\text{ср}}}$ form a right subtree (right subarray). The left subarray is interpreted as a new array. It similarly contains $j_{\text{ср. лев. 1/2}} = \left\lceil \frac{1}{2}\left(\left\lceil \frac{N}{2} \right\rceil - 1\right) \right\rceil = \left\lceil \frac{j_{\text{ср}} - 1}{2} \right\rceil$ root index. Here, $C_{j_{\text{ср. лев. 1/2}}}$ is the left-nearest descendant of the root of $C_{j_{\text{ср}}}$ tree. All components of the subarray to the left of $C_{j_{\text{ср. лев. 1/2}}}$ do not exceed $C_{j_{\text{ср. лев. 1/2}}}$; all components of the subarray on the right are not less than $C_{j_{\text{ср. лев. 1/2}}}$. Simultaneously, the root index of $j_{\text{ср. прав. 1/2}} = \left\lceil \frac{N}{2} \right\rceil + \left\lceil \frac{1}{2}\left(\left\lceil \frac{N}{2} \right\rceil - 1\right) \right\rceil = j_{\text{ср}} + \left\lceil \frac{j_{\text{ср}} - 1}{2} \right\rceil$ right subarray is determined. At this, $C_{j_{\text{ср. прав. 1/2}}}$ is the nearest descendant of the root of $C_{j_{\text{ср}}}$ tree. The process recursively resumes in each pair of the adjacent subarrays:

$$j_{\text{ср. лев. } 1/2^i,\ 1} = \left\lceil \frac{j_{\text{ср. лев. } 1/2^{i-1}} - 1}{2} \right\rceil,$$

$$j_{\text{ср. лев. } 1/2^i,\ 2} = j_{\text{ср. лев. } 1/2^{i-1}} + \left\lceil \frac{j_{\text{ср. лев. } 1/2^{i-1}} - 1}{2} \right\rceil,$$

$$j_{\text{ср. прав. } 1/2^i,1} = j_{\text{ср. прав. } 1/2^{i-1}} - \left\lceil \frac{j_{\text{ср. прав. } 1/2^{i-1}} - j_{\text{ср. прав. } 1/2^{i-2}} - 1}{2} \right\rceil,$$

$$j_{\text{ср. прав. } 1/2^i,\ 2} = j_{\text{ср. прав. } 1/2^{i-1}} + \left\lceil \frac{j_{\text{ср. прав. } 1/2^{i-1}} - j_{\text{ср. прав. } 1/2^{i-2}} - 1}{2} \right\rceil,\ i=1,\ 2,\ \ldots,\ \log_2 N.$$

As a result, all components of the lower level of the binary tree are formed in $O(1)$ time. The process can continue until $\log_2 N$ exhaustion of the levels of the binary tree.

The number of algorithm steps for constructing the binary tree in a parallel form is the sum of the sorting step and the step sequence when calculating the indices of the roots of subtrees. From here, $T(R) = \log_2 N\,\tilde{\tau} + \tau = O(\log_2 N)$, where R is the number of processor elements, τ is the time of binary comparison, and $\tilde{\tau}$ is the time for calculating one root index. R number of processors is determined by the maximum N parallel sorting of input elements, and then – through the calculation of indices with doubling by the number of tree levels. When calculating the indices, this number will not exceed $2^{\log_2 N-1} = N/2$, therefore the number of processors involved in sorting is sufficient. As a result, R will be less than $\frac{N^2-N}{2}$ [3]. Finally, the time complexity of the parallel algorithm for constructing the binary tree will be

$$T\left(\frac{N^2-N}{2}\right) = O(\log_2 N).$$

Example [3]. The binary tree for an array of 15 elements $X = (14,\ 9,\ 24,\ 7,\ 11,\ 20,\ 28,\ 3,\ 8,\ 10,\ 13,\ 17,\ 21,\ 25,\ 30)$ is constructed as follows.

The result of the sort is the array

$$C = (3,\ 7,\ 8,\ 9,\ 10,\ 11,\ 13,\ 14,\ 17,\ 20,\ 21,\ 24,\ 25,\ 28,\ 30).$$

The root of the binary tree is the medial element of C array: $j_{\text{ср}} = \left\lceil \frac{15}{2} \right\rceil = 8,\ C_8 = 14$. The left subarray has $j_{\text{ср. лев. } 1/2} = \left\lceil \frac{8-1}{2} \right\rceil = 4$ root, $C_4 = 9$ element is the root of the left subtree, which is the left-nearest descendant of $C_{j_{\text{ср}}}$ medial component. The right subarray has $j_{\text{ср. прав. } 1/2} = 8 + \left\lceil \frac{8-1}{2} \right\rceil = 12$ root, $C_{12} = 24$ element is the root of the right subtree and the right-nearest descendant of $C_{j_{\text{ср}}}$ root. Further, $j_{\text{ср. лев. } 1/4,1} = \left\lceil \frac{4-1}{2} \right\rceil = 2$, $C_2 = 7$ element is the root of the subtree on the left and the left-nearest descendant of the root of $C_{j_{\text{ср. лев. } 1/2}}$ subtree. In the right subtree, the root has $j_{\text{ср. лев. } 1/4,2} = 4 + \left\lceil \frac{4-1}{2} \right\rceil = 6$ number, $C_6 = 11$ element is the root of the right subtree and right-nearest child of $C_{j_{\text{ср. лев. } 1/2}}$.

Similarly, to the left of $C_{j_{\text{ср. прав. } 1/2}}$, $j_{\text{ср. прав. } 1/4,1} = 12 - \left\lceil \frac{12-8-1}{2} \right\rceil = 10$ root is determined, $C_{10} = 20$ element is the root of the left subtree from it and the left-nearest descendant of $C_{j_{\text{ср. прав. 1/2}}}$ subtree root. For the subarray, adjacent to the right one discussed above, the root has $j_{\text{ср. прав. }1/4,2}=12+\left\lceil\dfrac{12-8-1}{2}\right\rceil=14$ number, $C_{14}=28$ element is $C_{j_{\text{ср. прав. 1/2}}}$ right-nearest descendant and the right subtree root. The lower level of the tree will be formed by the descendants remaining to the left and to the right of each of four identified roots (Fig. 1):

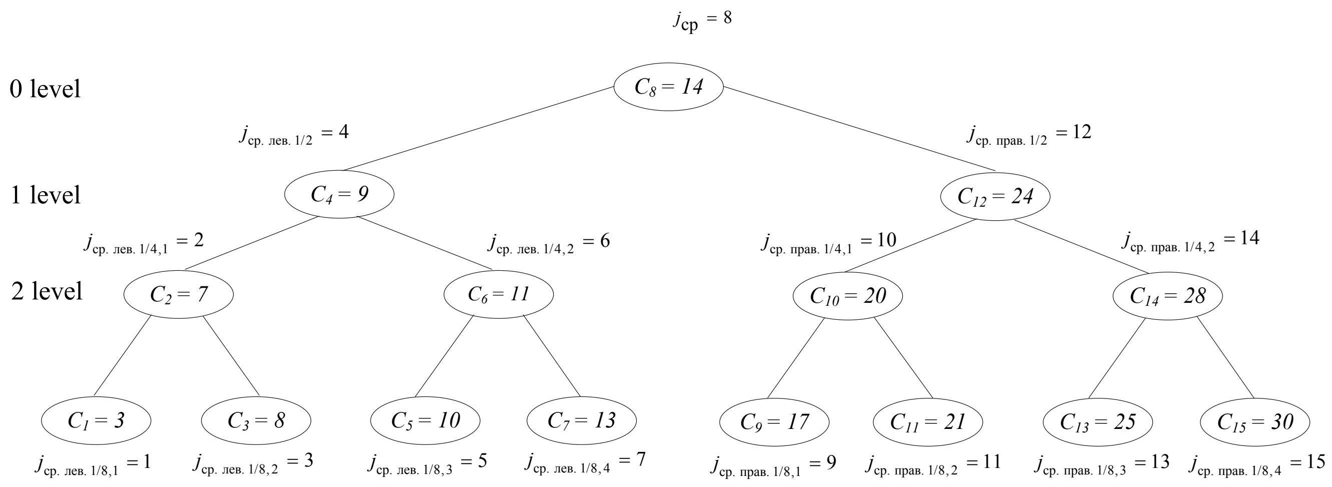
$j_{\text{ср}}=8$
0 level
$C_8=14$
$j_{\text{ср. лев. 1/2}}=4$
$j_{\text{ср. прав. 1/2}}=12$
1 level
$C_4=9$
$C_{12}=24$
$j_{\text{ср. лев. }1/4,1}=2$
$j_{\text{ср. лев. }1/4,2}=6$
$j_{\text{ср. прав. }1/4,1}=10$
$j_{\text{ср. прав. }1/4,2}=14$
2 level
$C_2=7$
$C_6=11$
$C_{10}=20$
$C_{14}=28$
$C_1=3$
$C_3=8$
$C_5=10$
$C_7=13$
$C_9=17$
$C_{11}=21$
$C_{13}=25$
$C_{15}=30$
$j_{\text{ср. лев. }1/8,1}=1$
$j_{\text{ср. лев. }1/8,2}=3$
$j_{\text{ср. лев. }1/8,3}=5$
$j_{\text{ср. лев. }1/8,4}=7$
$j_{\text{ср. прав. }1/8,1}=9$
$j_{\text{ср. прав. }1/8,2}=11$
$j_{\text{ср. прав. }1/8,3}=13$
$j_{\text{ср. прав. }1/8,4}=15$

Fig. 1. Example of constructing binary tree based on sorting

There is **Theorem 1** [3]. For a single-dimensional array of N components, a binary tree can be built in parallel using sorting with $T\left(\dfrac{N^2}{2}\right)=O(\log_2 N)$ time complexity.

The used sorting is stable; the binary tree is implied to be constructed with uniqueness. The indices of all medial components (all roots of subtrees) can be identified [3]. Considering this modification, all the indices from the above example for N subtree values can be calculated synchronously and mutually independently. This leads to a single estimate of the build time of the binary tree. For each specific N, all the values of the tree node indices can be calculated a priori and stored in the computer memory. With their help, the sorted components can be synchronously and mutually independently addressed to all the tree nodes. Formulas for calculating the node indices depend only on the total number of N input elements and are in no way dependent on their mutual arrangement after the stable sorting. To simplify memory addressing, the computed indices can be ordered at each level and arranged in ascending levels. Then, the entire population of the ordered node indices is read from N key. It only remains to arrange the sorted tree elements by the read-in addresses. Based on the above, there is

Theorem 2. For a single-dimensional array of N components, a binary tree can be built in parallel using sorting and prior calculation of indices with $T\left(\dfrac{N^2}{2}\right)=O(1)$ time complexity.

The following unified table contains the formal estimates of time complexity of sequential and parallel algorithms for constructing a binary tree versus the proposed algorithms.

Table 1

Comparative estimates of time complexity of sequential and parallel algorithms for constructing binary tree versus proposed algorithms

Binary tree algorithm	Algorithm time complexity	Acceleration when using unit time-complexity algorithm	Acceleration when using logarithmic time-complexity algorithm
Algorithm of A. Lagana and V. Kumar (2004) [5]	$\tilde{T}=O\left(N^{\frac{k+1}{k}}\right)$ [5]	$\frac{\tilde{T}}{T^{*}}=O\left(\frac{N^{\frac{k+1}{k}}}{1}\right)$	$\frac{\tilde{T}}{T}=O\left(\frac{N^{\frac{k+1}{k}}}{\log_2 N}\right)=O\left(N\ln 2\right)$
Algorithm of P. Chalermsook (2015) [6]	$\tilde{T}=O\left(N^{2}\right)$ [6]	$\frac{\tilde{T}}{T^{*}}=O\left(N^{2}\right)$	$\frac{\tilde{T}}{T}=O\left(\frac{N^{2}}{\log_2 N}\right)=O\left(N^{2}\ln 2\right)$
Polynomial algorithm (2016) [7]	$\tilde{T}=O\left(N^{3}\right)$ [7]	$\frac{\tilde{T}}{T^{*}}=O\left(N^{3}\right)$	$\frac{\tilde{T}}{T}=O\left(\frac{N^{3}}{\log_2 N}\right)=O\left(N^{3}\ln 2\right)$
"Left child – right sibling" algorithm (2014) [8]	$\tilde{T}=O\left(N^{2}\right)$ [8]	$\frac{\tilde{T}}{T^{*}}=O\left(N^{2}\right)$	$\frac{\tilde{T}}{T}=O\left(\frac{N^{2}}{\log_2 N}\right)=O\left(N^{2}\ln 2\right)$
Pattern-based algorithm (1991) [9]	$\tilde{T}=O\left(\lvert D\rvert\log_2 D\right)$ [9]	$\frac{\tilde{T}}{T^{*}}=O\left(\frac{\lvert D\rvert\log_2 D}{1}\right)$	$\frac{\tilde{T}}{T}=O\left(\frac{\lvert D\rvert\log_2 D}{\log_2 N}\right)$
The presented algorithm with logarithmic estimate of time complexity (2015) [3]	$T=O\left(\log_2 N\right)$ [3]	–	–
The presented algorithm with single estimate of time complexity (2015) [3]	$T^{*}=O(1)$	–	–

In Table 1: D is capacity of the template dictionary, N is the number of input elements of the binary tree, k is the dimension of the space in which sorting is performed.

The table shows that the proposed algorithm with a logarithmic estimate of time complexity abstractly improves estimates of the known algorithms. Minimum acceleration is achieved with respect to the algorithm from [5]: $\frac{\tilde{T}}{T}=O\left(N^{2}\ln 2\right)$, or $\frac{\tilde{T}}{T}=O\left(N\right)$; and maximum acceleration is achieved relative to the polynomial algorithm from [7]: $\frac{\tilde{T}}{T}=O\left(\frac{N^{3}}{\log_2 N}\right)$ or $\frac{\tilde{T}}{T}=O\left(N^{3}\right)$. Regarding the proposed algorithm with a single estimate of time-complexity, the evaluation of the known algorithms also improves. In this case, minimum acceleration is achieved relative to the algorithm from [5]: $\frac{\tilde{T}}{T^{*}}=O\left(N\right)$, and maximum acceleration is achieved with respect to the polynomial algorithm from [7]: $\frac{\tilde{T}}{T^{*}}=O\left(N^{3}\right)$.

Conclusion. The developed algorithms differ from the known techniques [5–7, 10, 11] of constructing a binary tree in that they use maximum parallel sorting to calculate the indices of the nodes. In this case, either a logarithmic number of steps is consumed by building a tree, or additional time is not spent at all, if the values of the indices are a priori calculated for all N values in some real boundaries and stored in the computer memory. The proposed parallel algorithm for constructing a binary tree can be used to organize efficient methods for dynamic processing of databases.

References

1. Romm, Ya.E., Chabanyuk, D.A. Sravnenie slov s edinichnoy vremennoy slozhnost'yu. [Comparison of words with the complexity of identity] Izvestiya SFedU. Engineering Sciences. 2014, no. 7 (156), pp. 230–238 (in Russian).

2. Romm, Ya.E. Parallel'naya sortirovka sliyaniem po matritsam sravneniy. II [Parallel sorting by merging on comparison matrices. II] Cybernetics and Systems Analysis, 1995, no. 4, pp. 13–37 (in Russian).

3. Romm, Y.E., Chabanyuk, D.A. Postroenie dvoichnogo dereva na osnove parallel'noy sortirovki. [Constructing binary tree based on parallel sorting algorithm.] Fundamental Research, 2015, vol. 8., no. 3, pp. 509–513 (in Russian).

4. Romm, Ya.E., Chabanyuk, D.A. Parallel'noe postroenie dvoichnogo dereva na osnove sortirovki. [Parallel construction of binary tree based on sorting algorithm.] Aspekty razvitiya nauki, obrazovaniya i modernizatsii promyshlennosti: mater. Vseross. nauchno-prakt. konf. [Aspects of development of science, education and industrial modernization: Proc. All-Russian Sci.-Pract. Conf.] Taganrog, 2017, vol. 1, pp. 77–84 (in Russian).

5. Laganà A., Kumar, A., Tan, V.C. Computational Science and Its Applications: Lecture Notes in Computer Science. Assisi: Springer Science & Business Media, 2004, 1044 p. – DOI: 10.1007/b98048

6. Chalermsook, P., Goswami, M., eds. 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS 2015). Piscataway, NJ: IEEE, 2015, pp. 410-423 – DOI: 10.1109/FOCS.2015.98

7. Gavrikov, A.V. T-neprivodimye rasshireniya dlya orientirovannykh binarnykh derev'yev. [T-irreducible extensions of directed binary trees.] Computer Sciences and Information Technologies, 2016, no. 6, pp. 123–125 – DOI 10.17223/20710410/34/6 (in Russian).

8. Gritsenko, N.S., Belov, Yu.S. Postroenie dvoichnogo dereva na osnove modifitsirovannoy skhemy khraneniya derev'yev obshchego vida «left child»-«right sibling» (LCRS). [Creation of a binary tree based on the modified storage diagram of general appearance trees "left child - right sibling" (LCRS).] Engineering Journal: Science and Innovation ,2014, no. 3, pp. 75–84 — DOI: 10.18698/2308-6033-2014-3-1281 (in Russian).

9. Amir, A., Farach, M., Adaptive dictionary matching. Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on. IEEE, 1991, pp. 760-766 – DOI: 10.1109/SFCS.1991.185445

10. Fischer, J., Heun, V. Theoretical and Practical Improvements on the RMQ-Problem, with Applications to LCA and LCE. Combinatorial Pattern Matching. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 4009, pp. 36-48.

11. Institute of Electrical and Electronics Engineers. Pattern-Avoiding Access in Binary Search Trees. Computer Society. 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS 2015), 2015, no. 56, pp. 410-423 - DOI: 10.1109/FOCS.2015.32

Received 21.01.2018
Submitted 24.01.2018
Scheduled in the issue 20.07.2018

Authors:

Romm, Yakov Ye.,
Head of the Information Technology Department,
Taganrog Chekhov Institute, Rostov State University of
Economics (RINH) branch (48, Initsiativnaya St.,
Taganrog, RF), Dr.Sci. (Eng.), professor,
ORCID: http://orcid.org/0000-0002-7251-2844
romm@list.ru

Chabanyuk, Denis A.,
associate professor of the Theoretical, General Physics and
Technologies Department, Taganrog Chekhov Institute,
Rostov State University of Economics (RINH) branch
(48, Initsiativnaya St., Taganrog, RF),
ORCID http://orcid.org/0000-0003-2972-0944
denchabanyuk@gmail.com