УДК 004

Shostak Anatoliy, Ph.D., Associate Professor
National Aerospace University «Kharkiv Aviation Institute»

ON MODIFICATION OF THE ALGORITHM FOR CONSTRUCTING AN AVL TREE

The AVL tree data structure is widely used in data processing system software as a height-balanced structure that provides logarithmic speed for search, insertion, and deletion operations [1-3].

Let there be a set of integers S of power N. It is necessary to construct an AVL tree based on S. Classically [1-3], the A1 algorithm for constructing an AVL tree is used for this task, and the construction of the tree depends on the order of the elements in S. In algorithm A1, after inserting the next node, the height balance of the nodes is checked and, if necessary, the tree is balanced in height by means of single and double left and right rotations of the subtrees. The result is an AVL tree with a height of approximately $\log_2 N$. The complexity of such an algorithm for constructing an AVL tree is $O(N\log N)$.

For the sequence SA consisting of elements (1, 2, 3, 4, 5, 6, 7), the AVL tree is shown in Fig. 1, and four left rotations were required to balance it in height.

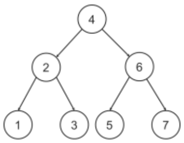

Fig. 1 – AVL tree for a sequence of elements (1, 2, 3, 4, 5, 6, 7)

For the sequence SB (4, 2, 6, 1, 3, 5, 7) of the same numbers, but in a different order, the AVL tree has the same appearance (Fig. 1), but does not require a single rotation during construction. That is, the order of the SB sequence is such that, as a result of inserting keys into the binary search tree, an AVL tree will ultimately be constructed without the need to check the balance of nodes and left and right rotations.

A distinctive feature of SB-type sequences is that the median of the entire sequence is in the first place, the median of the left subtree is in the second place, the median of the right subtree is in the third place, and so on. That is, if you preprocess the initial sequence before constructing the AVL tree and obtain an SB-type key sequence, the result will be the construction of an AVL tree without the need to check the balance of nodes after each insertion and perform left and right rotations.

It is proposed to use algorithm A2 to construct the AVL tree, which performs the following steps:

1) based on the set of integers S, obtain the ascendingly sorted sequence S1 (the complexity of such sorting can be *O*(N), for example, for the counting sort algorithm),

2) select the median of the sorted sequence S1 and place it in the first position in the sequence S2 — this will later be the root of the AVL tree being constructed (the complexity of searching for the median in this case is *O*(1)), the left part of S1 relative to the median forms the nodes of the left subtree of the AVL tree, and the right part of S1 relative to the median forms the nodes of the right subtree,

3) recursively continue selecting medians for the left and right parts of S1 and thus form a sequence S2 of N integers (the complexity of forming S2 is *O*(N)).

4) construct an AVL tree based on S2 (obviously, it is not necessary to check the balance of nodes and balance by height for S2).

The complexity of constructing an AVL tree using algorithm A2 is also *O*(NlogN), but algorithm A2 has a significantly lower multiplicative constant for the following reasons. Algorithm A2 uses a more efficient and optimised sorting operation compared to the more complex and slower operations of traversing the tree, checking its balance, recursive returns to update the tree balance, and the complex rotation logic of algorithm A1. Algorithm A2 also uses faster access to data in a dense array compared to access via links to tree nodes, which may be scattered across different memory locations, in algorithm A1.

References

1. Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., Clifford Stein. Introduction to algorithms: / Thomas H Cormen. – MIT Press, 2022. – 1312 pp. ISBN: 9780262046305

2. Brown Russell A. Comparative Performance of the AVL Tree and Three Variants of the Red-Black Tree. Software: Practice and Experience, 2025, Vol. 55(9). P. 1607-1615. https://doi.org/10.1002/spe.3437

3. Bounif L., Zegour D. Toward a Unique Representation for AVL and Red-Black Trees. Computación y Sistemas, 2019, Vol. 23, No. 2, P. 435–450. https://doi.org/10.13053/cys-23-2-2840