ISSN 2250-0987
Omprakash Deshmukh, UNIASCIT, Vol 3 (1), 2013, 333-339

OPTIMIZATION OF SPACE COMPLEXITY OF TREE TRAVERSAL

Omprakash Deshmukh[1], Mandakini Kaushik[2], Shikha Deshmukh[3]

[1]M.Tech.(SE) IIIT Gwalior, System Analyst, Netcracker, #5&43 HiTec City,Hyderabad, AP - 500081.
[2]M.Tech.(CSE) Scholar, Dept. of CSE,, Rungta College of Engg. & Tech., Bhilai – 490 024(C.G.), INDIA
[3]Msc(Microbiology), Vellore Institute of Technology, Vellore - 632 014

ABSTRACT— Binary search tree is a best-suited data structure for data storage and retrieval when entire tree could be accommodated in the primary memory. A binary search tree is a data structure that can support dynamic set operations i.e. Search, Minimum, Maximum, Predecessor, Successor, Insert and Delete. In basic operations, it takes time proportional to the height of the tree – O(h). A binary search tree is used to store ordered data to allow efficient queries and updates.
Our work has the basis of understanding of the various popular Binary search tree algorithms and introducing new algorithm with the optimization of space complexity of BST. The applications where primary memory is used and database is too large, it suffers from bottleneck problem of space. The minimum required space in O(1) helps to recover from this problem.
The traversal of search tree comes under the depth first search; the in-order traversal is most useful for application level programs because they provide sorted sequences. The main motivation behind trying to optimize the space complexity of search tree is to provide more independence from hardware requirement of memory. In other words, it provides hardware independence that comes under resource management.

Keywords- Pre-order, In- order, Post-order .

I. INTRODUCTION

Binary search tree is most basic, nonlinear data structure in computer science that can be defined as "a finite set of nodes that is either empty or consist of a root and two disjoint subsets called left and right sub-trees". A binary search tree is a tree with an ordering relationship between data in the nodes (i.e. all nodes with smaller data are in the left sub tree and all nodes with greater or equal data are in the right sub tree). The binary search tree (BST) is a data structure for holding information that allows rapid access according to some ordering key. It is a tree structure in which the nodes containing information can be connected to two sub trees. A binary search tree is used to store hierarchically ordered data to allow efficient queries and updates.

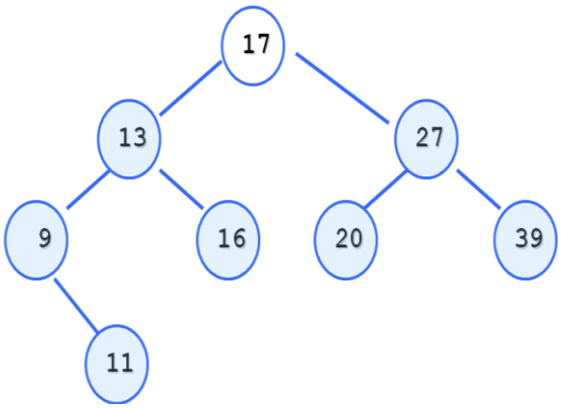

Figure 1: Example of Binary Search Tree

Complexity It is the analysis of the amount of memory and time, an algorithm requires to completion. Efficiency of an algorithm denotes the rate at which an algorithm solves a problem of size n. It is measured by the amount of resources it uses, the time and the space. There are two types of Complexity

Space complexity: RAM usage
Time complexity: CPU usage
Space complexity of an algorithm is the amount of memory it needs to run toCompletion .

An informal definition:
The (space) complexity of a program (for a given input) is the number of elementary objects that this program needs to store during its execution. This number is computed with respect to the size n of the input data.

A formal definition:
For an algorithm T and an input x, DSPACE(T; x) denotes the number of cells used during the (deterministic) computation T(x). We will note DSPACE(T) = O(f (n)) if DSPACE(T; x) = O(f (n)) with n = |x| (length of x). Note: DSPACE(T) is undefined whenever T(x) does not halt.

The components that are used for space complexity
Instruction Space
Environment Stack
Data Space.

In our work we are only in concern of optimizing a space complexity of BST. Our approach is based on modifying existing algorithm Building new procedure then introducing the New BST algorithm with minimum required space in $O(1)$.

II. BACKGROUNDS AND RELATED WORK

There is a large amount of work related to BST Algorithm. Maximum researchers have worked on the optimizing the Time complexity of bst algorithm. In our work we are only in concern of optimizing a space complexity of BST.

A binary tree is a special kind of tree, one that limits each node to no more than two children. A binary search tree, or BST, is a binary tree whose nodes are arranged such that for every node n, all of the nodes in n's left sub tree have a value less than n, and all nodes in n's right sub tree have a value greater than n. As we discussed, in the average case BSTs offer log2 n asymptotic time for inserts, deletes, and searches. (log2 n is often referred to as sub-linear because it outperforms linear asymptotic times.)

Binary Tree Traversal Methods

Pre-order

The root of the sub tree is processed first before going into the left then right sub tree (root, left, right).

```
Void preorder (TreeNode *CurrentNode)
{
Print CurrentNode->data;
preorder(CurrentNode->LeftChild);
preorder(CurrentNode->RightChild);
}
```

In-order

After the complete processing of the left sub tree the root is processed followed by the processing of the complete right sub tree (left, root, right).

```
Void inorder (TreeNode *CurrentNode)
{
Inorder(CurrentNode->LeftChild);
Print CurrentNode->data;
Inorder(CurrentNode->RightChild);
}
```

Post-order

The root is processed only after the complete processing of the left and right sub tree (left, right, root).

```
Void inorder (TreeNode *CurrentNode)
{
postorder(CurrentNode->LeftChild);
postorder(CurrentNode->RightChild);
Print CurrentNode->data;
}
```

Definition Of Tree Traversal

Traversing a tree means processing it in such a way, that each node is visited only once. The different types of traversing are

a. In-order traversal-yields infix form of expression.
b. Pre-order traversal-yields prefix from of expression.
c. Post-order traversal-yields postfix from of expression..

Predecessor and Successor

1. Successor of node x is the node y such that $key[y]$ is the smallest key greater than $key[x]$.
2. The successor of the largest key is NIL.
3. Search consists of two cases.
   - If node x has a non-empty right sub tree, then x's successor is the minimum in the right sub tree of x.
   - If node x has an empty right sub tree, then:
   - As long as we move to the left up the tree (move up through right children), we are visiting smaller keys.
   - x's successor y is the node that is the predecessor of x (x is the maximum in y's left sub tree).
   - In other words, x's successor y, is the lowest ancestor of x whose left child is also an ancestor of x or is x itself.

Pseudo-code for Successor

- Tree-Successor(x)
- 1. **if** $right[x] \neq NIL$
- 2. **then** return Tree-Minimum($right[x]$)
- 3. $y \leftarrow p[x]$
- 4. **while** $y \neq NIL$ **and** $x = right[y]$
- 5. **do** $x \leftarrow y$
- 6. $y \leftarrow p[y]$
- 7. **return** y

- Code for ***predecessor*** is symmetric.
- Running time: $O(h)$

Traditional Algorithm of Traversal

In-order Traversal

In-order after the complete processing of the left sub tree the root is processed followed by the processing of the complete right sub tree (left, root, right).

In-Order Traversal Algorithm

```
inOrder (TreeNode <T> n)
 {
                if (n != null)
```

```
        {
                inOrder(n.getLeft());
                visit(n)
                inOrder(n.getRight());
            }
}
```

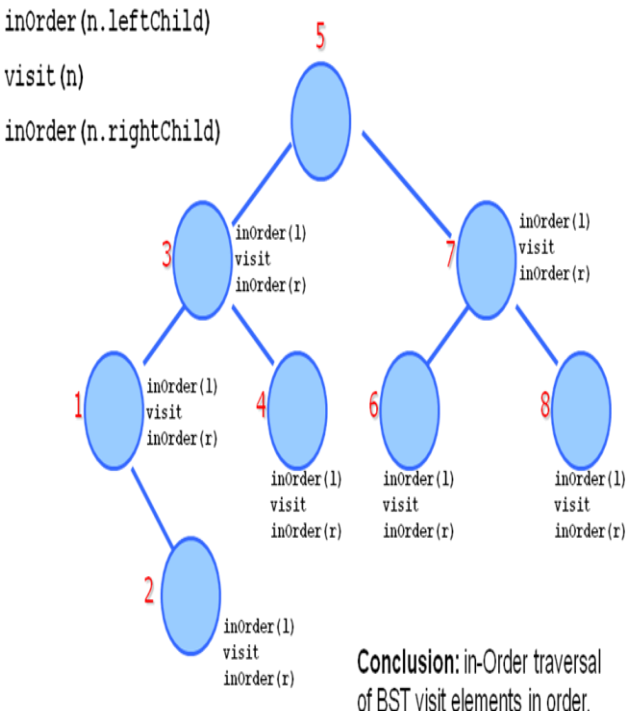

Figure 3.1 : BST in-order traversal

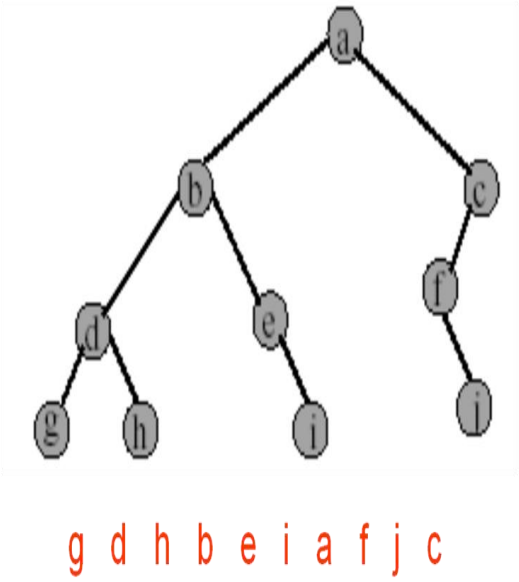

Figure 3.2 : In-order example (visit = print)

Pre-order Traversal
The root of the sub tree is processed first before going into the left then right sub tree (root, left, right).

Pre-Order Traversal Algorithm

```
preOrder (TreeNode <T> n)
 {
                if (n != null)
                {
                        visit(n);
                    inOrder(n.getLeft());
                        inOrder(n.getRight());
                    }

}
```

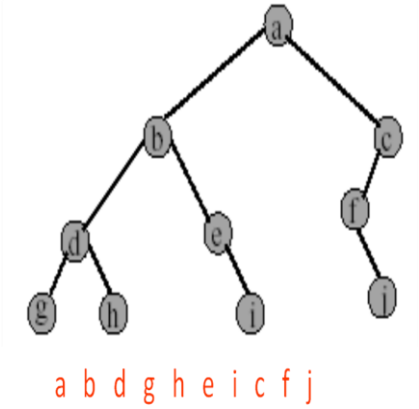

Figure 3.3 : Pre-order example (visit = print)

Post-order Traversal
The root is processed only after the complete processing of
the left and right sub tree (left, right, root).

Post-Order Traversal Algorithm

```
postOrder (TreeNode <T> n)
 {
                if (n != null)
                {
                        inOrder(n.getLeft());
                        inOrder(n.getRight());
                        visit(n)

                    }
 }
```

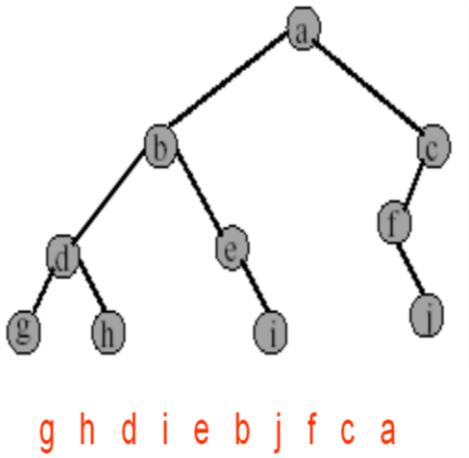

Figure 3.4 : Post-order example (visit = print)

The above algorithm can be summarized as follows:

- The space complexity of each of the three traversal algorithm is O(n)
- The time complexity of each of the three traversal algorithm is Θ(n)

Our work also implemented differential BST algorithm method. Having these inorder and preorder implementations of different algorithms in hand, our proposed method can be efficiently implemented for optimizing the size complexity of BST algorithm

III. PROBLEM STATEMENTS

The main challenge of these techniques is to optimize a space complexity. We will discuss Estimation Of Space Complexity for Existing In-order, Pre-order and Post-order Tree Traversal.
Every algorithm uses of computer resources to complete its task . The resources most important in relation to efficiency are central processor time and internal memory. It is always preferable to design algorithm that are economical in use of CPU time and memory because of the high computing resources.

Estimation Of Space Complexity Of In-order Tree Traversal

When the tree is traversed in in-order output is sorted collection of keys in ascending order. After traversal, the array is recursively partitioned into two parts left and right with each part differing in one key at most. Left partition creating left sub-tree and right partition right sub-tree . During each partition median has to be determined that becomes the root of left and right partition. Algorithm is not space efficient since it requires all the pointers to be copied into an array doubling the space requirement. Array needs contiguous memory that system may not provide sometimes particularly when data size is quite large. Since each node of the tree has to be visited, Algorithm runs in O(n) space complexity.

Estimation Of Space Complexity Of Pre-order Tree Traversal

Similar to the in-order traversal, the array is recursively partitioned into two parts left and right with each part differing in one key at most. Left partition creating left sub-tree and right partition right sub-tree. During each partition median has to be determined that becomes the root of left and right partition. The only difference is that it traverse left and right node before the root. Algorithm is not space efficient since it requires all the pointers to be copied into an array doubling the space requirement. Array needs contiguous memory that system may not provide sometimes particularly when data size is quite large. Since each node of the tree has to be visited, Algorithm needs O(n) space.

Estimation Of Space Complexity Of Post-order Tree Traversal

In post-order tree traversal, root is traversed at last. The remaining process of accessing all elements of tree is same. the array is recursively partitioned into two parts left and right with each part differing in one key at most. Left partition creating left sub-tree and right partition right sub-tree. During each partition median has to be determined that becomes the root of left and right partition. Since each node of the tree has to be visited, Algorithm takes O(n) space to execute .

We analyzed all the ways to traverse a binary search tree and the approaches to analyses space complexity in this paper

Problem Definition: A binary search tree is a data structure that can support dynamic set operations i.e. Search, Minimum, Maximum, Predecessor, Successor, Insert and Delete. In basic operations, it takes time proportional to the height of the tree – O(h). A binary search tree is used to store ordered data to allow efficient queries and updates.

Our work has the basis of understanding of the various popular Binary search tree algorithms and introducing new algorithm with the optimization of space complexity of BST. The applications where primary memory is used and database is too large, it suffers from bottleneck problem of space. The minimum required space in O(1) helps to recover from this problem.

IV. PROPOSED ALGORITHM

Theoretical Concept: The traditional approaches of binary search tree traversal either recursive or iterative uses O(n) space. But it is possible to traverse a tree without using n space complexity, if we are allow to temporarily using the available free link of node in tree. We can achieve all traversal in O(1) space.

As we know, it is easy to notice that in-order traversal is very simple to degenerate trees in which no node has a left child. Therefore, the usual three steps, LVR, turn into two steps, VR. No information need to be retained about the current status of the node being processed before traversing its left child, simply because there is no left child.

The idea is to achieve this traversal contains three steps. The in-order predecessor holds the pointer to the immediate predecessor node. Here, each root node is going to be visited twice. The same logic is applied to implement the in-order and pre-order traversal.

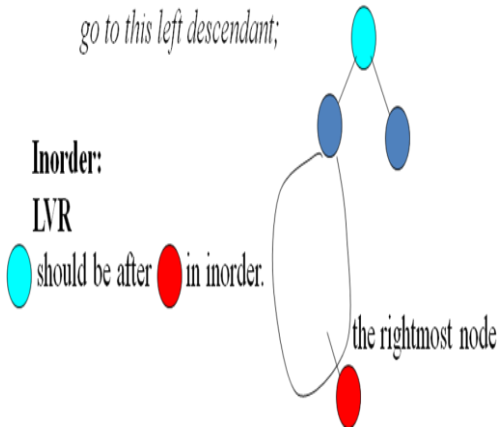

Figure 4.1 : Theoretical concept of In-order traversal.

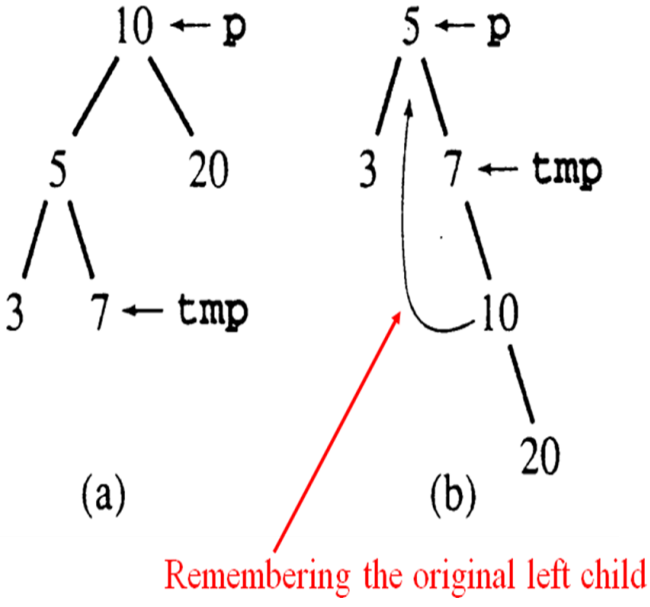

Figure 4.2 : Example of In-order traversal using Temp variable

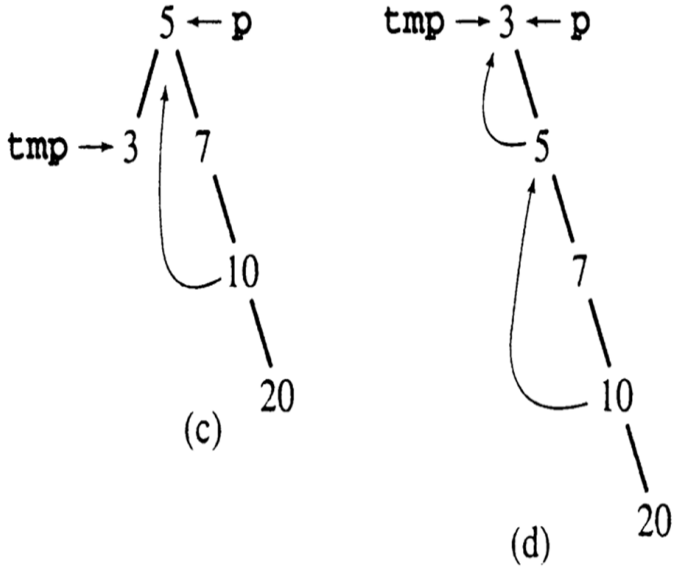

Figure 4.3 : Example of In-order traversal using Temp variable

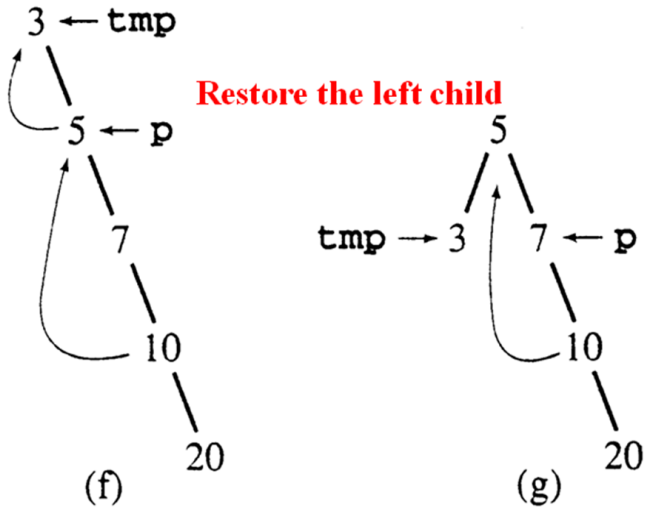

Figure 4.4 : Example of In-order traversal using Temp variable

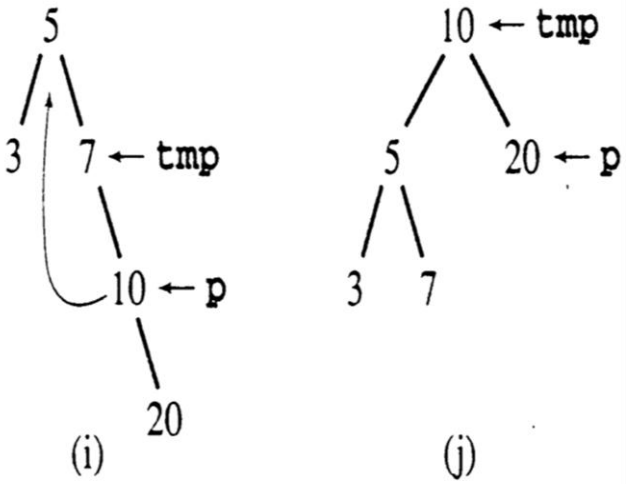

Figure 4.5 : Example of In-order traversal using Temp variable

Prposed Algorithm:

Algorithm 1 Pseudo code For In-order traversal

1. a is a root of tree.
2. **while** root is not NULL
   If root->left is NULL **Then**
   Print root->data
   Set root to root-> right
3. **Else**
   1. Set temp to root->left
   2. **While** temp->right is not NULL and temp->right is not equal to root
   3. set temp to temp->right
      End While
   4. **If** temp->right is NULL Then
      Set temp->right to root
      Set root to root->left
   5. **Else**
      Print root->data
      Set temp->right to NULL
      Set root to root->right
   6. **End If**
   7. **End If**
   8. **End While**

Algorithm 2 Pseudo code For Pre-order traversal

1. a is a root of tree.
2. **while** root is not NULL
3. **If** root->left is NULL **Then**
   1. Print root->data
   2. Set root to root->right
4. **Else**
   1. Set temp to root->left
   2. While temp->right is not NULL and temp->right is not equal to root
   3. set temp to temp->right
   4. **End While**
5. **If** temp->right is NULL **Then**
   1. Print root->data
   2. Set temp->right to root
   3. Set root to root->left
6. **Else**
   1. Set temp->right to NULL
   2. Set root to root->right
7. **End If**
8. **End If**
9. **End While**

The rendering process is a combination of modified QSplat and Surface splatting rendering algorithms. We traverse the preprocessed octree hierarchy using breadth first search (BFS) order as explained in Algorithm3. A node in the hierarchy is rendered when its projected area is less than a threshold or if it is a leaf node. The basic surface splatting idea can be used while displaying surfels.

V. RESULT AND DISCUSSION

We provide different input values to our program, which is implemented in C language. The code is working successfully for all the input data sets. It also takes the memory space 1 for in-order and pre-order tree traversal, which was the main motive.

As we are taking the available free link to point its in-order predecessor, no extra memory is occupied. Here one temporary variable is used, which is dynamically allocated. When the tree traversal is performed the left child of root node holds value of root through the temp pointer. Again root is visited to print its value, in case of in-order. Similarly, the re-order traversal uses this free link to visit again the root after traversal of left sub tree. Thus, the tree traversal with optimal memory made possible.

The analysis of algorithm for tree traversal shows the space complexity in O (1), which is successfully achieved.

VI. CONCLUSION

We analysed all the ways to traverse a binary search tree and the approaches to analyse space complexity in this paper. We are trying to achieve the optimal search tree traversal using a free link of node in tree. However , this may change the tree structure temporarily which need to be restored before the traversal is finished.

The main motivation of this project is the library writers who provide the inbuilt tree traversal

function within the package. The optimization of space complexity will provide the hardware independence and an efficient resource management in term of less memory requirements. The other application area of binary search tree are dictionaries and priority queues.

The only limitation of this project is that it is not applicable to multithreaded environment because of the temporary change in the basic structure of binary search tree while traversal is being performed.

VII. REFERENCES

[1]. Suri pushpa, prasad vinod . binary search tree balancing methods: a critical study., ijcsns international journal of computer science and network security, vol.7 no.8, august 2007.
[2]. Deaconu , optimal time and space complexity algorithm for construction of all binary trees from pre-order and Post-order traversals., the 7th balkan conference on operational Research.bacor 05.constanta, may 2005, romania.
[3]. Din iasi (serie noua), ed. Univ. .al. I. Cuza., tomul iv; a space and time efficient algorithm for constructing compressed suffix Arrays , grant r-252-000-119-112 o.h. Ibarra and l. Zhang (eds.): cocoon 2002.
[4]. e. Mäkinen (2000), .constructing a binary tree efficiently from its traversals., int. J. Comput. Math. 75, no.2, 143-147;
[5]. donald e. Knuth, the art of computer programming, volume 3: (2nd ed.) Sorting and searching, addison wesley longman publishing co., inc., redwood city, ca, 1998
[6]. donald e. Knuth, the art of computer programming, volume 1 (3rd ed.): fundamental algorithms, addison wesley longman publishing co., inc., redwood city, ca, 1997
[7]. r. K. Ahuja, t. L. Magnanti, j. B. Orlin (1993), .network flows: theory, algoritms and applications., prentice hall.
[8]. cormen, thomas h., charles e. Leiserson, and ronald l. Rivest. "introduction to algorithms." mit press. 1990.
[9]. Andersson, s. Carlsson (1990), .construction of a tree from its traversals in Optimal time and space. , inf. Process. Lett. 34, no.1, 21-25;
[10]. E.Mäkinen (1989), .constructing a binary tree from its traversals., bit 29, no.3, 572 575;