Array, Linked List, and Binary Search Tree Comparison

MAJA SAREVSKA
Faculty of Informatics, European University Skopje,
NORTH MACEDONIA

Abstract: - In this paper, I present the comparison of necessary average programming steps for element search and deletion in basic data structures, like arrays, linked lists, and binary search trees. It is well known that an array is a very convenient data structure for accessing elements via indices, while it is complicated for element deletion, as it requires rearranging all the rest elements. Sorted arrays are very convenient for binary element search. On the other side linked list is a data structure that requires only a few steps for adding and deleting an element. I quantitatively confirm that the binary search tree is efficient both for binary search and for element deletion. This analysis may be used for the education of undergraduate and graduate students in computer science.

Key-Words: - Array, linked list, binary search tree, binary search, element deletion, search algorithm.

Received: March 1, 2024. Revised: April 16, 2024. Accepted: May 22, 2024. Published: June 24, 2024.

1 Introduction

The most important aspects of any programming language are Data Structures (DS). Data are organized and stored in DS to be efficiently used for data operations. DS are arranged data in a particular way, saved in the memory so it can be retrieved to be used later, [1].

An array DS is a basic concept in programming, it is a collection of items of the same data type stored in contiguous memory locations, [2]. This DS is efficiently used in programming for manipulating and organizing data with access to any array element using indices. Another fundamental DS in programming is a Linked List (LL), it is made of a set of nodes, where each node is represented with data and reference or link to the next node, [2]. This DS efficiently adds and deletes elements in the LL. A Binary Tree (BT) is a tree DS where each node can have at most two children, and these two nodes are referred to as the left child and the right child. BTs have many applications in computer science, like data storage and retrieval. Also, they can be used to implement algorithms such as searching, sorting, and graph algorithms, [3]. A Binary Search Tree (BST) is a special type of binary tree in which the left child has a value less than the node's value and the right child has a value greater than the node's value. This property provides efficient application of various data operations like deleting, searching, or inserting elements in the tree and it is called the BST property, [4], [5].

The goal of this paper is to show exactly this property of BST quantitatively. Namely, BST can search elements efficiently like in a sorted array, and at the same time BST can efficiently delete and insert elements same efficiently like in LL. This quantity is represented by program steps in the programming language C. Section 2 defines illustratively and with programming code the appropriate DSs. Section 3 represents the comparison between BST and Array. Section 4 presents the comparison between BST and LL. And finally, in Section 5 some concluding remarks are given.

2 Array, Linked List, and Binary Search Tree in the Programming Language C

As I explained an array is a set of items of the same data type, stored in contiguous memory locations that may be accessed efficiently with indices, [4]. If the array is sorted then applying the appropriate algorithm we can do an element search in a very efficient and fast way. Figure 1 presents the array illustratively. Figure 1a presents the element deletion. First, the element should be found, then removed from the array, and afterward, all upper elements must be moved down to one place.

LL DS is a collection of elements called nodes, where each node is represented with data and reference or link to the next node. In this DS we can efficiently add and delete elements, [4]. Figure 2 illustrates the LL DS. Figure 2a presents the illustration of element deletion and the formation of new links while deleting the element.

LL DS is very convenient for adding and deleting a node [6], [7], in our analysis I will focus only on element deletion, although the conclusion may be easily driven in the case of adding an element.

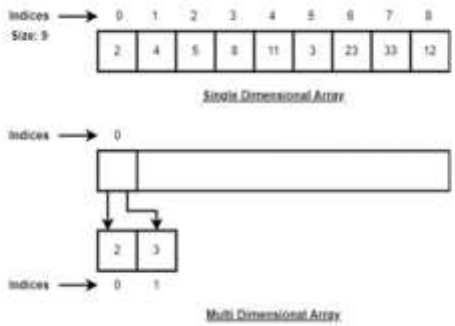

Fig. 1: Array DS, illustratively and programming code for definition in programming language C

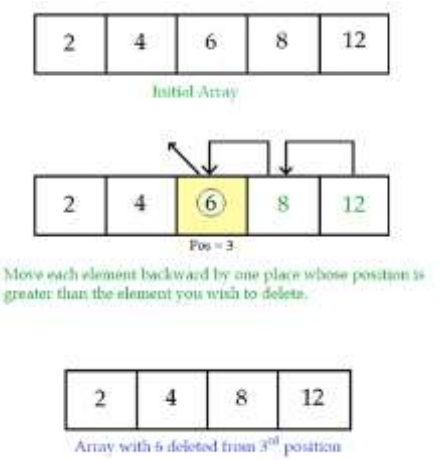

Fig. 1a: Array DS, element deletion, [8]

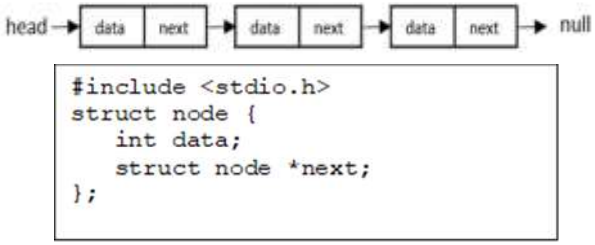

Fig. 2: LL DS, illustratively and programming code for definition in programming language C

Fig. 2a: LL DS, element deletion

BST is a special type of BT, [9]. [10], where the value of the left child is less than the value of the parent node and the value of the right child is greater than the value of the parent node, [4]. Figure 3 illustrates the BST DS. Figure 3a presents the element deletion in the BST and node rearranging after element removal.

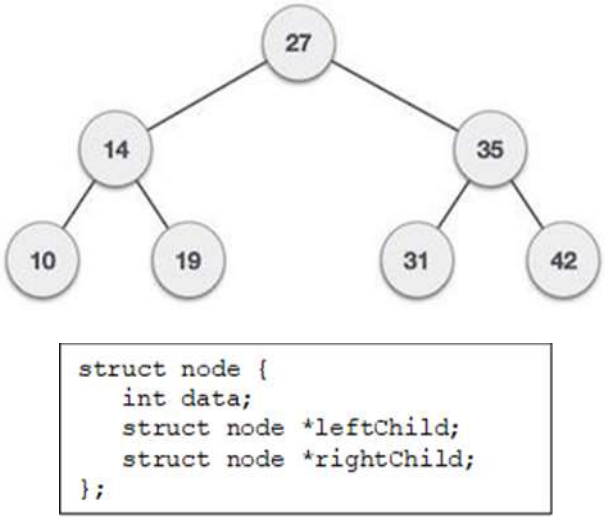

Fig. 3: BST DS, illustratively and programming code for definition in programming language C

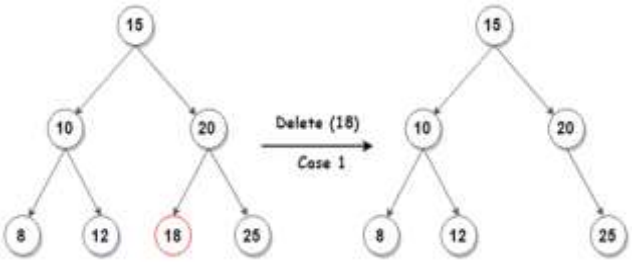

Fig. 3a: BST DS, element deletion, [4]

3 Comparison: Array and Binary Search Tree

The simulation experiment is done on the following data example, sorted array: {1,2,3,4,6,7,9,11,12,14,15,16,17,19,33,34,43,45,55, 66}. For the array, I use the Binary Search algorithm and I count the steps in the program. For BST I shuffle the elements, build the tree, and search for the appropriate element, and I also count the programming steps. I make thousands of trials and estimate the mean number of steps. Then I compare the steps, by presenting them in the chart in Figure 4.

For example, we may notice that the number of programming steps to find the element 66 in the array is 5 while for the BST is 3. Overall the values are comparable.

For the programming code in C, the reader is kindly asked to contact the author. In the program, I define function shuffle, which shuffles the elements in the sorted array. This is done because in general the items in the sequence are random when I build the LL or BST. Then I define a function find(item), that does the binary search for the element in the sorted array. Namely, if the element that should be found is greater than the middle element then I continue the search in the upper sub-half array. If not in the lower one. Iteratively I repeat this procedure while I find the element. Then I define the function insert(data), to build the BST by adding the elements (nodes) one by one. I define the function search(data). If the element that should be found is smaller than the node value I continue to search in the left sub-tree, else I search in the right sub-tree. I repeat this iteratively until I find the element. Then I define the main() function where I do the binary search, count the procedure steps while array binary search, shuffle the elements, build the BST, do the element search in the BST, and count the procedure steps.

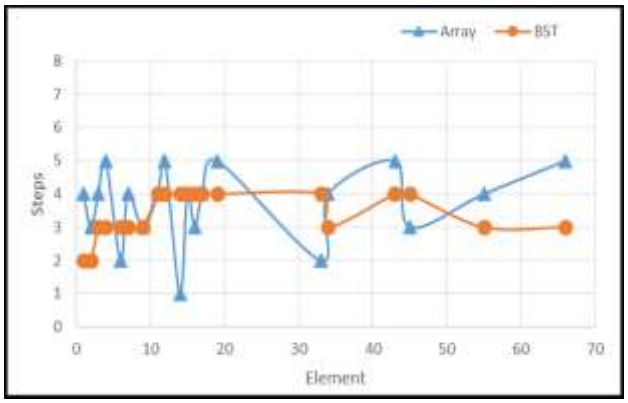

Fig. 4: Steps comparison between Array and the BST

Another simulation experiment is done on the following data example, sorted array: {1,2,3,4,6,7,9,11,12,14,15,16,17,19,33}. I decreased the number of elements. For the array, I use the Binary Search algorithm and I count the steps in the program. For BST I shuffle the elements, build the tree, and search for the appropriate element, and I also count the programming steps. I make thousands of trials and estimate the mean number of steps. Then I compare the steps, by presenting them in the chart in Figure 5.

For example, we may notice that the number of programming steps to find the element 33 in the array is 4 while for the BST is 3. Overall the values are comparable. As it is expected there is a decreased number of steps and the length of data space decreases. Also, there is an influence on the value of the data samples, but that's for future analysis.

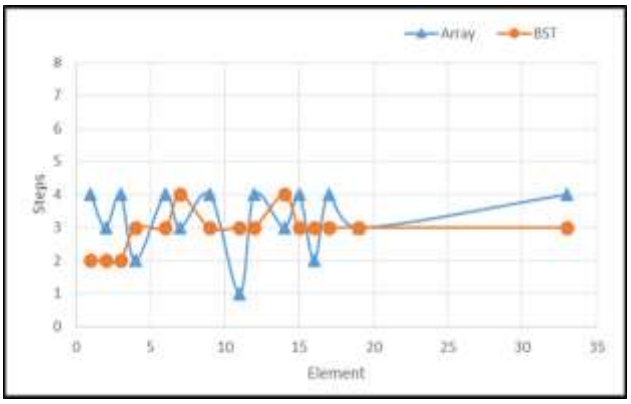

Fig. 5: Steps comparison between Array and the BST

4 Comparison: Linked List and Binary Search Tree

I again use the same data sample of 20 elements: {1,2,3,4,6,7,9,11,12,14,15,16,17,19,33,34,43,45,55, 66}. I shuffle the elements and form the LL and the BST. In both cases, I search for the appropriate element and delete it. I do this trial a thousand times and estimate the mean value of the programming steps. The comparison is presented in Figure 6.

For example, we may see that the average number of programming steps for LL is around 9 while for the BST is around 5. The better performance of the BST is thanks to the faster data search compared to the LL.

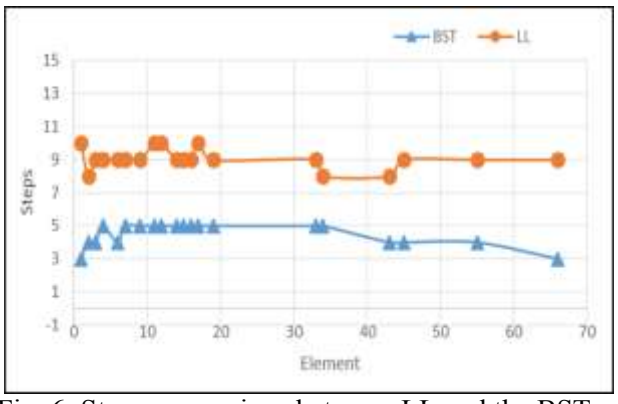

Fig. 6: Steps comparison between LL and the BST

For the programming code in C, the reader is kindly asked to contact the author. In the program, I define function shuffle, which shuffles the elements in the sorted array. This is done because in general the items in the sequence are random when we build the LL or BST. I define the function insert(data), to

build the BST by adding the elements (nodes) one by one. I define the function search(data). If the element that should be found is smaller than the node value I continue to search in the left sub-tree, else I search in the right sub-tree. I repeat this iteratively until I find the element. Then I define a function minValueNode, which is necessary when deleting the node. The node with minimal value in the left sub-tree should replace the deleted node, and other nodes should be rearranged. I define the function deleteNode for node deletion. I defined the function insertFirst(data) to build the LL by adding the nodes one by one. Next, I define the function delete to delete a node from the LL and reorganize the links. Then I define the main() function where I shuffle the elements, build the BST and LL, do the element search in the BST, and delete the desired element, do the element search in the LL, and delete the desired element, and I count the procedure steps.

5 Conclusions

In computer science, it is widely known that a sorted array is efficient for element search while complicated for element adding and deleting. On the other side, LL is very convenient for element adding and deleting. For a given sample of a data sequence of 20 elements, I confirmed that BST is performing binary element search as efficiently as a sorted array. Also for the same data sample, I quantitatively confirmed that the BST is the same efficient in element deletion as LL. This analysis may be used for the education of undergraduate and graduate students in computer science.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

The author wrote, reviewed, and edited the content as needed and she has not utilized, artificial intelligence (AI) tools. The author takes full responsibility for the content of the publication.

References:

[1] Rubi Dhankhar, Sapna Kamra, Vishal Jangra, ”Tree concept in data structure”, *2014 IJIRT*, Vol. 1, Issue 7, ISSN: 2349-6002.

[2] Sthuti J, Namith C, Shanthanu Nagesh, “Data Structures and its Applications in C”, *International Research Journal of Engineering and Technology (IRJET)*, Vol. 8 Issue 4, Apr 2021

[3] Dimitrios Samoladas; Christos Karras; Aristeidis Karras; Leonidas Theodorakopoulos; Spyros Sioutas, “Tree Data Structures and Efficient Indexing Techniques for Big Data Management: A Comprehensive Study”. *PCI'22: Proceedings of the 26th Pan-Hellenic Conference on Informatics*, November 2022, pp. 123–132.

[4] Tutorials Points. Data Structures and Algorithms (DSA) Tutorial. https://www.tutorialspoint.com/data_structures_algorithms/index.htm (Accessed Date: March 1, 2024).

[5] Data Structures Tutorial, [Online]. https://www.geeksforgeeks.org/data-structures/, (Accessed Date: March 1, 20024).

[6] Elshad Karimov, *Linked Lists*, Chapter in Data Structures and Algorithms in Swift, pp. 41-54 Apress , 2020.

[7] D. VarshaaA. Keerthana DeviM. Sujithra, Fundamentals of Data Structures: Stacks, Queues, Linked Lists, and Graphs, *Chapter n Advanced Applications of Python Data Structures and Algorithms* (pp.1-34), IGI Global, 2023.

[8] Techie Delight, [Online]. https://www.techiedelight.com/ (Accessed Date: March 1, 20024).

[9] Travis Gagie. New Ways to Construct Binary Search Trees, Conference: Algorithms and Computation, *14th International Symposium, ISAAC 2003*, Kyoto, Japan, December 15-17, 2003.

[10] Roberto De Prisco, Alfredo De Santis, On binary search trees, *Information Processing Letters*, Vol. 45, Issue 5, 2 April 1993, pp. 249-253.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare.