A Sorting based Algorithm for the Construction of Balanced Search Tree Automatically for smaller elements and with minimum of one Rotation for Greater Elements from BST

S. Muthusundari
Research Scholar, Dept of CSE, Sathyabama University
Chennai, India
e-mail: nellailath@yahoo.co.in

Dr. R. M. Suresh
Principal, Sri Muthukumaran Institute of Technology,
Chennai, India
e-mail: rmsuresh@hotmail.com

Abstract

Tree is a best data structure for data storage and retrieval of data whenever it could be accommodated in the memory. At the same time, this is true only when the tree is height-balanced and lesser depth from the root. In this paper, we propose a sorting based new algorithm to construct the Balanced search tree from Binary Search Tree with minimum of one rotation for the given elements n >14. If the given elements n < 14 then the algorithm automatically constructs the Balanced Search tree without needs any rotations. To maintain the tree in shape and depth, we apply two strategies in the input data. The first one is to apply sorting on the given input data. And the second one is to find the multiples of two positions on the sorted input data. Then, we compare the 3 positions of multiples of two and rewrite it by descending order and repeat this for the entire elements and the rest of the positions also on the sorted data. After, a new input data is formed. Then construct the Binary search tree on the given input data. At last, we will find the output as; a height balanced BST (AVL) with lesser depth from the root for the smaller data such as N < 14, for the greater element N >14 , it requires one rotation from the BST., and the search cost is minimum as possible. In this paper, few case studies have been carried out and analyzed in terms of height and space requirement. Hence, the height of the output BST, normally obtain by maximum height as N/2 or (N/2) – 3.

Key words : Binary Search Tree; depth; divide and conquer; sorting; height balanced, Rotation

1. Introduction

Binary search tree is a, nonlinear data structure in computer science that can be defined as a finite set of nodes that is either empty or consists of a root and two sub nodes as left and right sub-trees. In general situation, we expect the tree to be of minimal height that is possible only when the tree is height balanced. With a *n* node random binary search tree search time increases by *O(lg(n))* as size of input grows. A binary search tree requires approximately 1.36*(lg (n))* comparisons if keys are inserted in random order. There are many methods to rebalance a binary search tree by dynamic rebalancing, and global (static) rebalancing and self rebalancing. These methods are having advantages and disadvantages. Dynamic methods to create a balanced binary search tree have been around since Adel'son-Velskii & Landis [1] proposed the AVL Tree. Dynamic rebalancing method helps to improve the tree in shape and optimal. Readjustment is needed when search is required. So, this could be achieved by various ways. Many researchers proposed solutions to improve the efficiency of the binary search tree by its optimal shape and minimize the search cost.

Binary Search tree is a binary tree in which each internal node *x* stores an element such that the element stored in the left subtree is less than the root value and the right subtree is greater than the root value. This technique is called binary-search-tree property. The basic operations is based on a binary search tree is proportional to the height of the tree. For a complete binary tree with node n, such operations runs in O(lg *n*) worst-case time. The height of the Binary Search Tree is defined as the number of links from the root node to the last node. For the interested task to achieve, in this paper we present a new sorting based divide and conquer approach to

rebalance the binary search tree and to minimize the search cost by decrease its depth of the height almost by N/2 -3.

1.1 AVL Algorithm

To describe AVL trees we need the concept of tree height, which we define as the maximal length of a path from the root to a leaf. AVL tree is a binary search tree with a balanced condition. The balanced factors of the AVL tree are as {-1, 0, 1}.

Height Invariant [1] **.** At any node in the AVL tree, the heights of the left and right sub trees differ by at most 1. If the height of the binary search tree is not satisfied in the balanced factor, then it needs to be readjusted to balance the height of the binary search tree. To perform, we require the rotations operations. Based on the condition situated we need to perform single or double rotations to readjust the height of the binary search tree. In addition to that, sometimes, double rotation may be required making whole process a complex task and leads to make difficult. To avoid this situation in readjusting, we need to move alternate suggestions to construct automatic avl tree without using the notation operations to be performed. The following sections deal with more to obtain the above situations.

2. Related Work

Martin and Ness [5] implemented an algorithm that reconstructs a binary search tree with n nodes by repetitively subdividing n by 2 and uses the results as guidance to step through the framework of a perfectly balanced tree, i.e.,for each node, the number of nodes in its left subtree and right subtree differ by 1 at most. They were carried out by an in-order traversal procedure to provide relative node positions for the pointer restructuring, which makes them into proper places in the balanced tree structure. In this algorithm, a stack is required to save pointers during traversal.

Bentley [1] developed an algorithm to produce perfectly balanced trees. In this algorithm, nodes are passed as a set or a linked list to the balancing procedure which finds the median element of the set of nodes as root and splits the set into two subsets, each forming a balanced subtree at the next level. This process continues until there are no more elements to split.

Another algorithm that needs no extra storage was given by Day [4]. The input tree is required to be right threaded with negative backtrack pointers to allow stackless traversal**.** Day's algorithm consists of two phases. In the first phase, tree is traversed in in-order to visit every node, creating a backbone (linked list). The final linked list was a sorted list of items as in-order traversal results in sorted output. In second phase, degenerated tree is transformed into a balanced tree by performing a series of left rotations. Day used a threaded tree concept also to traverse the tree therefore some additional information has to be maintained in each node to distinguish a thread pointer from actual pointer. Day's algorithm is not very demanding as far as space requirement is concern during execution time. However, time complexity of algorithm is O(n), still it is very efficient algorithm if perfect balance is not required.

Sleater and Tarjan's [6] introduced a new concept for splay trees, are as efficient as balanced trees when total running time is the major concern rather than the cost of individual operation. To reduce total access time frequently accessed items should always be near to the root. They invented a simple method of restructuring a tree so that after each access accessed item moves closer to the root assuming, this item will be accessed again in near future (Principle of locality). According to Principle of locality any item that is accessed now, it is highly likely that same item will be accessed in near future.

The following table 1 summarizes the performances of the various existing algorithms in terms of the method used.

Sno	Algorithm	Method
1	Martin & ness	In order tree traversal & repeated dividing
2	Bentley	Median & split method
3	Day	Threaded tree &linked list
4	Sleater & Tarjan	Principle of locality

Table 1. Different Existing algorithms and its methods

3. Proposed Method

We present a new sorting based algorithm for the automatic construction of balanced tree form BST. Our proposed algorithm consists of three functions like

- Sort the given input data
- Find the multiples of two positions and compare the elements
- Construct the BST

3.1 Algorithm

The algorithm for the proposed sorting based divide and conquer approach method is given below in the fig. 1.

Algorithm
Step 1. Get the n elements as input
Step 2. Sort the n elements and store in a new array.
Step 3. Find the multiples of 2 in the given array.
Step 4. If the given element is below 7, compare the multiples of 2 by two array positions and shuffle in descending, or compare it by 3 positions and shuffle.
Step 5. Then repeat the step 4 for the non multiples of two positions.
Step 6. Construct BST
Step 7: If the given element n > 14, then it requires only one rotation to perform Balanced Serach Tree.

Fig. 1. Algorithm

3.2 Case Study I

Step 1 : Consider the elements, n = 14

12 45 67 23 11 10 9 7 4 3 2 1 55 76

Step: 2 Sort the elements

1 2 3 4 7 9 10 11 12 23 45 55 67 76

Step : 3 and step 4 & 5

Find the multiples of two. And check N > 7 so compare by 3 positions in multiples of 2

1 2 3 4 7 9 10 11 12 23 45 55 67 76

Positions a[2],a[4] and a[6] to be compared, and write it as descending order. Now we get,

9 4 2

Again positions a[8],a[10] and a[12] to be compared, and write it as descending order. Now we get, 55 23 11

Then a[14] . hence we get 76.

Then write the multiples of 2 first and followed by rest of the positions also compared by same as step 4.

For the rest of the positions in the array, we get

1 3 7 10 12 45 67

By repeating the step 4 we get the positions to be compared as 3.

7 3 1 45 12 10 67

Now combine the entire elements we get,

9 4 2 55 23 11 76 7 3 1 45 12 10 67

Step : 6 Construct the BST. The BST is in the following fig 2.

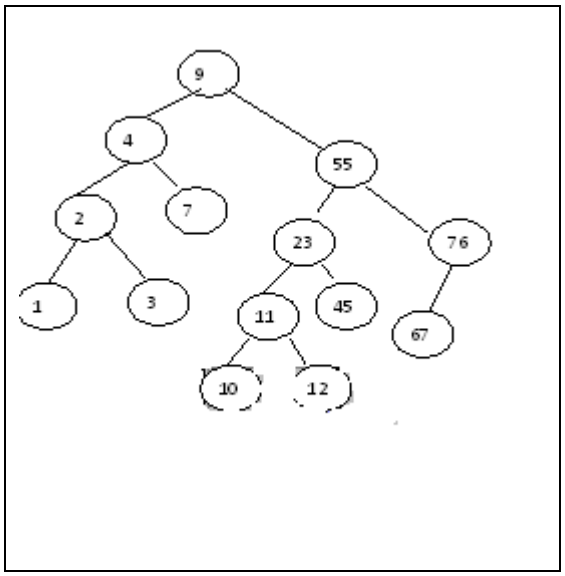

Fig. 2. Automaitc Balanced BST with Sorting Based Algorithm for N < 14

4. Comparison Of The Construction Of The Original Bst With Proposed Method

Let us consider the BST construction for the original given input data for the above case study example.

12 45 67 23 11 10 9 7 4 3 2 1 55 76

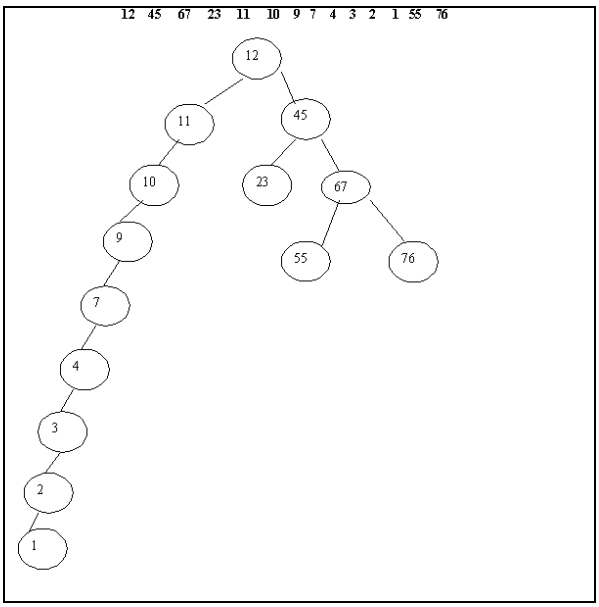

Fig. 3. BST with the original given input Data

Based on the above figure 3 the BST of the original given input data, the proposed method is automatically frame a Balanced BST and the depth of the root is 3. Incase, of the figure 2, the depth of the root value is 8. And it is not a balanced tree. Hence the proposed method is optimal to construct the Balanced BST.

4.1 **Case Study II**

Step : 1 Consider the elements n = 23

21 22 23 10 8 1 11 12 13 14 20 19 18 17 16 15 24 26 25 27 28 30 29

Step : 2 Sort the elements

1 8 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Step : 3 and step 4 & 5

Find the multiples of two, and check N > 7 so compare by 3 positions in multiples of 2

1 8 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Now we get,

13 11 8 19 17 15 25 23 21 29 27 12 10 1 18 16 14 24 22 20 30 28 26

Step : 6 Construct the BST

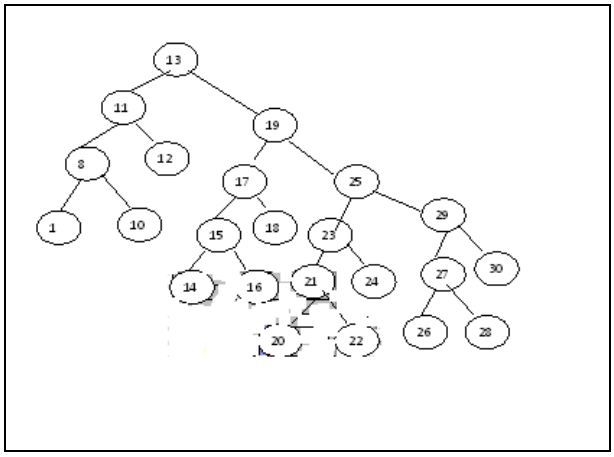

Fig. 4 BST Based on Sorting algorithm for N>14

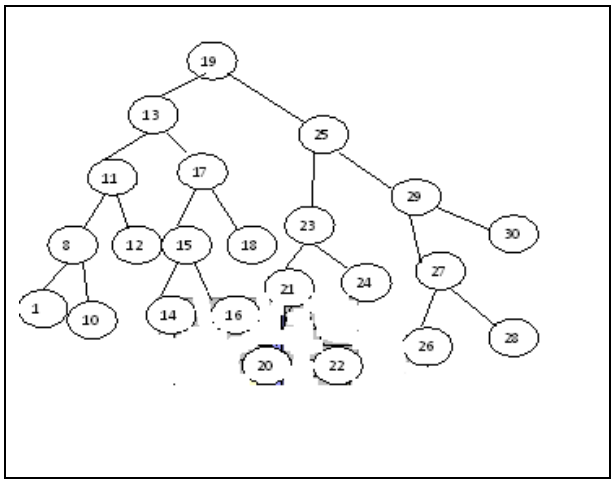

Fig. 5 Balanced Search Tree with One Rotation for N >14 elements, based on Algorithm

Step 7: N > 14, hence it requires only one rotation, and now the result of the Balanced Search Tree is given below.

5. Performance Of Search Cost Comparisons With Bst And Proposed Method

Given a sorted array of N elements of search keys and an array of frequency counts, are given where frequency is the number of searches to *keys*. Construct a binary search tree of all keys , then the total cost of all the searches is calculated.

Let us assume, the cost of a BST node is level of that node will multiply by its frequency. We assume the level of root is 1.

Consider the above case study I example with frequency value.

Input Data 12 45 67 23 11 10 9 7 4 3 2 1 55 76

Frequency 23 12 3 5 34 8 10 4 6 7 9 20 11 25

Sno	Key Value	Search Cost for Original BST		Search Cost for Proposed Method	
		Frequency	Search Cost	Frequency	Search Cost
1	12	1 * 23	23	5 * 23	115
2	45	2 * 12	24	4 * 12	48
3	67	3 * 3	9	4 * 3	12
4	23	3 * 5	15	3 * 5	15
5	11	2 * 34	68	4 * 34	136
6	10	3 * 8	24	5 * 8	40
7	9	4 * 10	40	1 * 10	10
8	7	5 * 4	20	3 * 4	12
9	4	6 * 6	36	2 * 6	12
10	3	7 * 7	49	4* 7	28
11	2	8 * 9	72	3 * 9	27
12	1	9 * 20	180	4 * 20	80
13	55	4 * 11	44	2* 11	22
14	76	4 * 25	100	3* 25	75
		Total	704		632

Table 2 Comparison of Search Cost for all the keys of Original BST with Proposed Method

For the original BST of the given input data the search cost is calculated as given in the table 2.

6. Result Analysis

Based on the comparisons were made, on the above case study example, the proposed method Sorting based algorithm approach is improved the search p
erformance is about 15 % as minimum total cost. The search cost for all the keys of original BST with Proposed method is given in the table 2 for the analysis.

The Results are analyzed by C Compiler. Based on the experiments results the proposed Sorting based algorithm is required only one rotation for constructing the Balanced Search Tree for the given elements n > 14, and for n < 14 it does not require any rotations, automatically the algorithm constructs the Balanced Search Tree.

To Study the performance of the proposed algorithm the code was executed 25 times and by the average the minimum of rotations required is given below in the table 3.

No.Of Data	Minimum no. of Rotations by Existing Method	Minimum no of rotations required by Proposed Sorting based algorithm
5	**1**	**0**
7	**2**	**0**
14	**6**	**0**
23	**11**	**1**
35	**17**	**1**
55	**28**	**1**

Table 3. Comparisons of Rotations

Hence the proposed sorting based algorithm automatically constructs the balanced search tree for the given elements n <14, and for n > 14 it needs only one rotation to construct the Balanced search tree. Based on the analysis, the number of rotations for the .given n elements is n/2 +2 by the existing approach for the proposed sorting based algorithm it takes n/2 -2.The height of the root is also lesser, and the **maximum height is obtained as N/2 or (N/2) – 3.**

7. Conclusion

In this paper, we have analyzed the performance of the proposed Sorting based algorithm for the original construction of BST. The result shows the new method is to perform minimum of 15% to minimize the search cost and increases the search process, and also it is automatically constructs the Balanced BST without using any rotations and for some values it requires only one rotation, and almost all the key values much be nearer to

the root. Hence, the depth or height of the BST is also comes much smaller. The proposed sorting based algorithm is normally suits for all the given elements where the number of elements is even. Some random values of the odd set it does not suit. In future enhancement, the modified algorithm will refine the problem that makes to suit. Almost the height of the root it takes randomly of given N elements maximum of N/2 or (N/2) – 3.

References

[1] S. Muthusundari and R. M. Suresh, " Sorting Based Divide and Conquer Approach Leads to the Automatic Construction of Balanced Tree from BST", National Conference on Advances in Computing and Technology(NCACT-2013), 15th March – 2013,Published by CIIT.

[2] S. Muthusundari and R. M. Suresh, "An Enhanced Bubble Sorting algorithm with Divide and Conquer Approach Leads to the Construction of Balanced Search Tree ", CiiTInternational Journal of Artificial Intelligent systems and Machine Learning,Vol 5, No 4, April 2013, pg no, 189 – 192,0974 – 9667.

[3] Adel'son-Vel'skii, G.M., and Landis, E.M., 1962, An Algorithm for the Organization of information. Soviet Mathematics Doklady, 3, pp.1259–1263.

[4] John Michael Robson. Constant bounds on the moment of the height of binary search trees. Theoretical Computer Science, 276(1–2):435–444, 2002.

[5] Chang, H., and Iyengar S.S., July 1984, Efficient Algorithms To Globally Balance a Binary Search Tree Communication of the ACM 27, 8, pp. 695-702.

[6] Day, A. C., 1976, Balancing a Binary Tree, Computer Journal, XIX, pp. 360-361.

[7] Martin, W.A., and Ness, D.N., Feb 1972, Optimal Binary Trees Grown with a Sorting Algorithm. Communication of the ACM 15, 2, pp. 88-93.

[8] Sleator, D.D., and Tarjon R. E., July 1985, Self-Adjusting Binary Search Trees. Journal of The ACM, 32(3), pp. 652-686.

[9] Stout, F., and Bette, L. W., September 1986, Tree Rebalancing in Optimal Time and Space Communication of the ACM, Vol. 29, No. 9, pp. 902- 908.

[10] Bodo Manthey and R¨udiger Reischuk. Smoothed analysis of binary search trees. Theoretical Computer Science, 378(3):292–315, 2007.

[11] Donald E. Knuth. Sorting and Searching, volume 3 of The Art of Computer Programming. Addison-Wesley, 2nd edition, 1998.

[12] Bruce Reed. The height of a random binary search tree. Journal of the ACM, 50(3):306–332, 2003.

[13] Bruce Reed. The height of a random binary search tree, Journal of the ACM, 50(3):306–332, 2003.