Insertion and Deletion on Binary Search Tree using Modified Insert Delete Pair: An Empirical Study

[1] **Suri Pushpa,** [2] **Prasad Vinod,** [3] **Jilani Abdul Khader**

[1] Dept. of Computer Science, Kurukshetra University, Haryana, India

[2] Dept. of Technology, Majan University, Sultanate of Oman.

[3] Dept. of Computer Science, Nizwa University, Sultanate of Oman

Summary

Recently, a new version of the insert-delete pair has been proposed that maintains a random binary search tree in such a way that all the grandparents in the tree always have both of their sub-trees full. For a tree with *'n'* nodes, if such an arrangement is made, it is straightforward that even an arbitrary sequence of insertion and deletion would not cause the tree to grow beyond *n/2.* Compare this with conventional insert delete algorithms where a tree may grow up to the height of *n-1* reducing search to be sequential. Lesser the height of the tree, lesser would be its internal path-length, and hence faster the search would be. To study the behavior of the Binary Search tree with proposed insert delete pair, we have simulated a binary search tree with 1024 nodes. In this article we provide some empirical results to show that the proposed insert-delete pair maintains the tree in better shape, with considerable reduction in the internal path-length.

Keywords:

Tree Balancing, Binary Search Tree, Tree Path-Length

1. Introduction

Binary search trees are used in computer science for rapid data storage and retrieval. With an ideally arranged BST with *n* nodes, most of the tree related operations require time that is not more than *lg(n)*. That means effort required to perform an operation on a BST grows logarithmically as size of the input grows. Despite of its wide popularity binary search tree has few serious problems. One of the major problems with binary search tree is its *topology*. The BST topology depends upon the order with which data is added or deleted. That means if input is not in random order the tree becomes lengthier on one side, reducing the search to be sequential. For optimal results the tree has to be wider and flatter in shape. In other words, tree height has to be minimal so that resulting tree could become bushy. To maintain the tree in better shape many algorithms have been proposed over the years. Some of them are [1], [2], [3], [5], and [6]. Since many versions of the insert-delete algorithms exist, we would like to touch upon the conventional insert-delete algorithms. In the conventional insert algorithm, a key is inserted by first comparing the key with the root, and then based on the comparison a left or right path is chosen, repeating the process until the key is either inserted or discarded (if duplicate). For the conventional delete algorithm, two variations symmetric and asymmetric deletion exist. In asymmetric deletion, a leaf node is deleted simply, a node with one son is deleted by replacing the node by its son, and a node with two sub-trees is deleted by replacing the node by its in-order successor or predecessor. In symmetric deletion, to maintain the tree balance, successor and predecessor replaces the node alternatively.

2. Internal Path Length of the Tree

When analyzing the performance of a binary search tree we require few parameters like its height or internal path length. Though, these parameters are interrelated but sometimes give better performance evaluation when analyzed independently. The height of the tree is the longest path from the root to a leaf node. The internal path length *(IPL)* of a tree is the sum of the depths of all nodes in the tree. Root of the tree has depth zero, and every son in the tree has a depth that is one more than its parent. This means that *IPL* of the tree would be minimum when every node in the tree has minimal depth. This is possible only when the tree is height balanced. For an average case analysis of a binary search tree, the internal path length is an important parameter. Using *IPL* of the tree, average number of comparisons required to perform search, insert or delete can be computed. For a 'n' node random binary search tree, the internal path length I_n, and the average number of comparison required for a successful search C_n are related as $I_n = n(C_n - 1)$. This formula clearly indicates that average search time is directly related to the path length of the tree. Knuth [7] has given a formula that relates the height of a '*n*' node random binary search tree h_n and the number of comparison required for a successful search C_n as $C_n = 2(1 + 1/n)\, h_n - 3$. For a perfect balanced tree with *n* nodes, and height *h,* we have a relation *n* =

Manuscript received December 5, 2007
Manuscript revised December 20, 2007

2^{h+1} *-1*. In this case path length of the tree would be minimum and given by $I_{\min} = (n+1)\, lg(n+1)/lg2 - 2n$. .

Eppinger [4] gathered some empirical evidences to study the behavior of a binary search tree when operated with a series of conventional insert-delete algorithms. His experiment suggests that applying many insertions and asymmetric deletions causes a tree to become more unbalanced. However, insertions with symmetric deletions preserve the balance of the tree. His results suggest that during the first few insertions and deletions, *IPL* of the tree decreases, and after some critical point the *IPL* starts increasing, and eventually leveling of after approximately n^2 insertions and deletions.

3. Proposed Insert-Delete Algorithms

Height and *IPL* of the tree increases if too many nodes in the tree have just one son. In addition to that, a tree may loose its balance following two many insertions and deletions, resulting in increased *IPL* of the tree. This happens because conventional insert or delete algorithms do not take care of the “balance factor” of the tree. In our previous work [8], we have proposed another version of the insert-delete algorithms to maintain a random binary search tree dynamically. Without applying any complex rebalancing technique, or using considerable amount of space, proposed insert-delete pair maintains the tree in such a way that every grandfather in the tree always has two sons (provided that the grandfather exist). If such an arrangement were made, any sequence of insertion and deletion would keep the tree height <= *n/2*, and not *n-1*, that is what possible with the conventional insertion and deletion. A comparative study of the conventional and proposed insert algorithms (without deletions), given in [8], shows that for a random input, the proposed insert algorithm produces a tree with 20% to 30% reduction in the height, forcing the average number of comparisons required for a successful search to go down by 15% to 20%. Because of the space constraints, in our previous work [8], we could not provide any result to show that what happens to the tree when proposed insertion and deletion algorithms are applied together. Choosing path length of the tree *(IPL)* as a parameter, we have investigated that what happens to the tree when proposed insert delete algorithms are applied together.

4. Methodology

To measure the behavior of the proposed insert-delete algorithms [10] we have compared two binary search trees for a same sequence of corresponding insert-delete pairs. Comparing binary trees for arbitrary sequence of insertion and deletion is difficult since two trees may not have same number of nodes after each sequence. Initially, two trees of *‘n’* nodes are created using the conventional and proposed insert algorithms. To create initial binary search trees, random number generator generates numbers until *‘n’* unique numbers have been generated (ignoring duplicates). Each number (key) is then supplied to the conventional and proposed insert algorithms to produce conventional and proposed binary search trees. While inserting a key, if the key is already present in the tree, the key is discarded, and another random number is generated. Once initial trees are ready, their internal path length is calculated. Following that, a sequence of corresponding delete-insert pair is applied to each of the tree in such a way that the number of nodes in the trees always remains constant. To delete a node from the tree, a random number (key) is generated until produced number is found in the tree, and then, the key is deleted from both of the trees. In essence, each delete-insert pair ensures that a key is actually deleted and inserted in every sequence, so that the number of nodes in the trees always remains *‘n’*. For both of the trees, after each sequence, the path length is recorded. For the purpose of simulation, two initial trees with 1024 nodes are created using the conventional and proposed insert algorithms. Following this, each tree is operated by a series of 10,000 corresponding delete-insert pairs to gather following results.

5. Results

The graphs in figure 1 show the internal path lengths of the two trees against the number of insert-delete pairs. Three sample runs are provided. Thick line represents the internal path length of the tree operated by conventional insert-delete pair and vice versa. Maximum difference in the path lengths of the two trees was found to be in the very beginning (when no deletion was applied). As given sample runs show, internal path lengths of the two initial trees (proposed and conventional) were found to be (9431, 10694), (9471, 11913), and (9879, 11188). These results show that there is a considerable reduction in the path length of the tree created by proposed insert algorithm. Once ID pairs are applied, gap between the path lengths starts decreasing. As number of ID pairs increases, conventional tree seems to be loosing its path length (as observed by Eppinger), while path length of the proposed tree almost remains same. One could guess that loosing internal path length is always a desirable property, even in that case; tree operated by proposed ID pair always remains better in shape with lesser internal path length. Algorithms are tested for 10,000 ID pairs, and made to run under Borland C++ compiler 5.5, maximum random number that system could have generated was 32767.

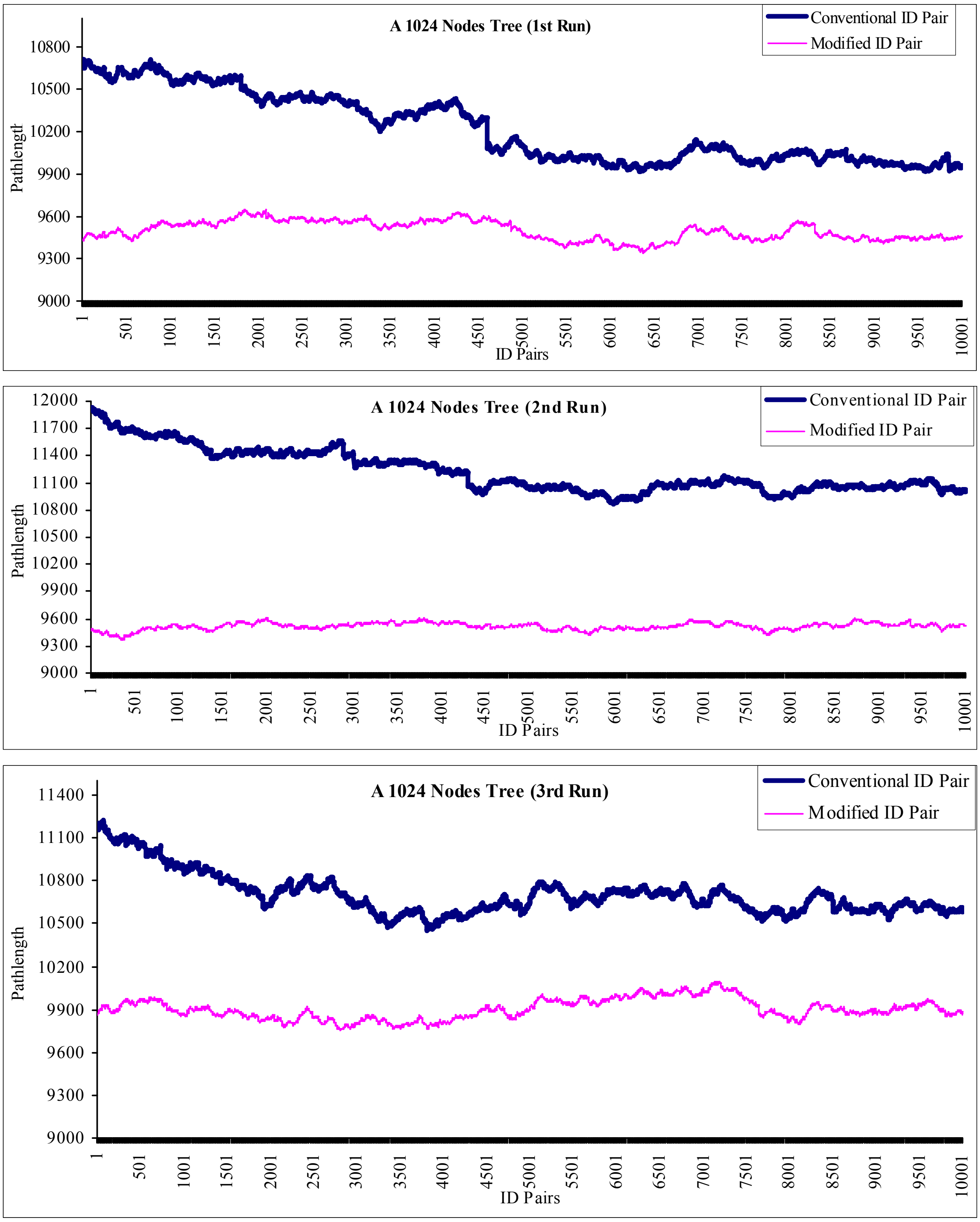
A 1024 Nodes Tree (1st Run)
Conventional ID Pair
Modified ID Pair
Pathlength
ID Pairs
10800
10500
10200
9900
9600
9300
9000
1
501
1001
1501
2001
2501
3001
3501
4001
4501
5001
5501
6001
6501
7001
7501
8001
8501
9001
9501
10001
A 1024 Nodes Tree (2nd Run)
Conventional ID Pair
Modified ID Pair
Pathlength
ID Pairs
12000
11700
11400
11100
10800
10500
10200
9900
9600
9300
9000
A 1024 Nodes Tree (3rd Run)
Conventional ID Pair
Modified ID Pair
Pathlength
ID Pairs
11400
11100
10800
10500
10200
9900
9600
9300
9000

Figure 1

References.

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

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

[3] Day, A. C. Balancing a Binary Tree. Computer Journal. 19, 1976. PP 360-361.

[4] Eppinger, J. L. An Empirical Study of Insertion and Deletion in Binary Search Trees. Commun.of the ACM. 26, 1983. PP 663-669.

[5] Chang, H. and Iyengar, S. S. Efficient Algorithms to Globally Balance a Binary Search Tree. Commun. of the ACM. 27, 1984. PP 695-702.

[6] Stout, F. and Bette, L. W. Tree Rebalancing in Optimal Time and Space. Commun. of the ACM. 29, 1986. PP 902-908.

[7] Knuth, D. E. The Art of Computer Programming. Vol. 3, Searching and Sorting, Pearson Education Asia, 1999.

[8] Vinod, P. Suri, P. and Maple, C. Maintaining a Binary Search Tree Dynamically. Proceedings of the 10th International Conference on Information Visualization. London, UK. 5-7th July 2006. PP 483-488.

Dr. Pushpa Suri is a reader in the department of computer science and applications at Kurukshetra University Haryana India. She has supervised a number of PhD students. She has published a number of research papers in national and international journals and conference proceedings.

Vinod Prasad has Master's degrees in Computer Science and Mathematics. At present, he is pursuing his PhD in Computer Science. His area of research is Algorithms and Data Structure where he is working on Binary search tree data structures. Vinod has published and presented a number of papers in national and international journals, and conference proceedings.

Abdul Khader Jilani has Master's degrees in Computer Science. At present, he is pursuing his PhD in Computer Science. A.K.Jilani presently working in the department of computer science ,College of Arts and science , University of Nizwa, Sultanate of Oman. He has published a number of research papers in national and international journals and conference proceedings.