Upgraded Tango Tree to solve the Dictionary Problem and its Applications

V S Anirudha Kaki
Department of Computer Science and Engineering
Motilal Nehru Institute of Engineering and Technology
Allahabad, India.

Suneeta Agarwal
Department of Computer Science and Engineering
Motilal Nehru Institute of Engineering and Technology
Allahabad, India.

***Abstract*—In 1989 Wilber [2] conjecture a lower bound O (log n) for a query using any balanced existing binary search tree with static data. Later in 2004 Demaine et al.., [1] Came up with new lower bound O (loglogn) called interleave lower bound and he also claimed that these two lower bound O (log n) and O (loglogn) acts as a good tight intervals for any binary search tree.**

Tango tree was recently introduced by Demaine *et al*., [1], having O (loglogn) - competitive ratio. Tango tree [1] supports only lookup (search) operations, where most of the online algorithms (like dictionary problem, a cache problem, adaptive data compression, etc.,) need additions and removals of collections (key, value) too, which are not supported by tango trees [1].

In this paper, we propose a new upgraded version of tango tree which supports addition and removal of collections (key, value) dynamically without knowing the sequence before hand in O (loglogn) time. We show run-time analysis with experimental results. We also show theoretically how these algorithms achieve O (loglogn) -competitive dynamic interleave bound.

Keywords- Binary search tree; Red-black tree; Splay tree; Tango tree; O(1)-competitive conjecture.

I. INTRODUCTION

A dictionary is a basic data structure that is capable of storing (storing objects in sorted order based on keys as a string or integers which appears at most once, composed of a collection of (key, value) pairs) and retrieving information. This abstract data type is one of the most important structures in computer sciences.
Operations associated with this data type allow:

- The addition of pairs to the collection.
- The removal of pairs from the collection.
- The modification of the values of existing pairs.
- The lookup of the value associated with a particular key.

Binary search tree (BST) provides data structures which efficiently support all dictionary operations, with flexibility and adaptability to a large number of purposes.

In the last decade many different BST data structures have been developed which perform these data type dictionary operations (associative array, map, etc.) In Ω (log n) time, where n is no of elements in the tree. For a specific access sequence (x_1, x_2 ..... x_m) there can be any BST algorithm which required a total running time of O (mlogn). This disparity however does not rule out the possibility of having an instance optimal BST, through *competitive analysis*: for any access sequence (x), let OPT (X) denotes the minimal cost for executing the access sequence X by any BST algorithm to serve it. A given BST algorithm A has a competitive ration of α for all sequences of operations X, we have $COST_A <= \alpha COST_{OPT}$ (X).

In 1985, Sleater and Tarjan [3] developed a BST called *splay tree*, as "Best BST" which they conjecture to be O (1) -competitive. However, they are later known to be O (log n) competitive ratio. So, this led researcher's looking at the efficiency of BST's on different input sequence has grown out of this conjecture and to search for a BST which is optimal (or close to optimal) on any sequence of the search which has grown out of this conjecture. Given splay trees have a number of remarkable properties like , including the Balance Theorem [3], the Static Optimality Theorem [3], the Static Finger Theorem [3], The Working Set Theorem [3], the Scanning Theorem [5], the Sequential Access Theorem [5, 7], and the Dynamic Finger Theorem [10, 11]. They are a natural candidate for the dynamic optimality. The Dynamic Optimality Conjecture [3] states that for any sequence of accesses, the cost of splay trees on that sequence of accesses is within a constant factor of any other binary search tree algorithm for processing that sequence of accesses. That is, it states that splay trees are O (log n) competitive [3] for some constant. However, over decades of research, still the conjecture is open for research.

After several attempts solve it for about 20 years, this situation was recently improved by Demaine *et al*., [1] Developed an O (loglogn) -competitive BST structure, called *tango tree,* suggested searching for alternative binary search tree algorithm which has a small, but non constant competitive factor, they proposed tango. This was the first major improvement in O (loglogn) competitive ratio over previous (trivial) competitive ratio of O (logn) upper bound, after Slater and Tarjan [3] progress on dynamic optimality.

In the journal version of seminal papers on tango trees Demaine et al., [1] stated tango as a static tree with dynamic search which doesn't support insertion and deletion. In this paper we came up with the upgraded version of tango as dynamic tango tree which support both insertion and deletion in O (loglogn) time.

The rest of the paper is organized as follows: In section II, we discuss BST model and stated its lower bound on OPT (X) developed for competitive ratio. In section III, we defined the data structure of the tango tree. In section IV, we discuss that tango algorithm. In section V, we upgrade

Proc. of the Second International Conference on Advances in Computing, Control and Communication (CCN)
Editor In Chief Dr. R. K. Singh.

ISBN: 978-981-07-2579-2

tango tree called the dynamic tango tree which support both insertion and deletion over the same tango tree.

II. BST MODEL

In this paper, we used the BST model which was defined by Wilber [2]. Every node keeps a key from a totally ordered universe which obeys in-order at any node. Basically an online BST data structure argument each node in a BST with additional data. Each unit cost operation changes the pointer referentially of the node to that new node pointer. The access algorithm's choice of the next operation to perform is a function of the data and augmented data stored in the node currently pointed to. In general the behavior of the algorithm depends on the past. The amount of argument information should be as small as possible. For example, red-black trees use one bit [4] and splay tree do not use any [3].

A. *Optimality*

Consider any BST data structure that executes any particular access sequence X optimally [9]. Let the cost per operation made by fastest BST data Structure for X is denoted as OPT (X). In other words any fastest offline BST algorithm can execute X in OPT (X), as the model doesn't restrict its next move from a BST access algorithm because it depends on the future access to come. Standard balanced BST establishes that OPT (X) = O (mlogn). In some closer of access sequence X (x_1, x_2 ..... x_m) Wilber [2] proved OPT (X) = Ω (mlogn).

A BST data structure is said to be dynamically optimal, only by the execution of all the sequences of X in time O (OPT (X)). Apart from this, it is still a query that such a data structure exists. BST data Structure is c-competitive if it executes all sequence X in time at most cOPT(X). Finally this model has O (log log n) augmented bits.

The main objective of the line of research is to design a dynamically optimal O(1)-competitive) online BST data structure that uses O(1) augmented bits per node. The result would be a single, asymptotically best BST data structure.

B. *Interleave Lower Bound*

Given any BST data structure (T_o) and a m-element access sequence X (x_1, x_2 ..... x_m), time taken by T_o to execute X, depends only on X. This lower bound is said to be interleaved lower bound. Interleave lower bound is a slight variation of Wilber [2] first bound OPT (T_o, X). This lower bound is also similar to lower bound that follows from partial sums in the semi group model [8, 10].

Let's consider a perfect binary tree P on the keys {1, 2, 3 ….. n} for accessing the sequence X, we denoted interleave lower bound as IB(P,X). For each node y in P, left region of y is defined as y plus all nodes of the y's left subtree in P and similarly right region of y is defined as all nodes of the y's right subtree in P. Next, label each access in the restricted sequence X over P. We label each access sequence x_i as left/right based on its region over y, discarding all the access outside y's in P. Amount of interleave IB(P,X,V) is no of the times the label switches between left and right. Interleave bound IB (X) is the sum of these interleave amounts over all node y in P.

Theorem 1: IB (X) /2-and are a lower bound on OPT (X). The cost of the optimal offline BST that serves access sequence X.

$$OPT(T_o,X) >= IB(P,X)/2\text{-}O(n) + m$$

III. THE TANGO TREE DATA STRUCTURE

Consider a static balanced BST of height O (logn) which is made on a set of n nodes, refer this tree as reference tree P. The depth of any node in P is atmost 2log(n+1) as we consider P as balanced tree. At any point of time each node in P has a preferred child may be its left child or right child. In other words we can state that preferred child of node y in P would be left node or right node based on the last access to a node within y's subtree in P was in left region of y (including y) or right region of y respectively. After executing the access sequence X (x_1, x_2 ..... x_m), the state of P is changed to P_m over augmented perfect binary search tree. The structure of the reference tree P is static (we are going to modify it to support insertion and deletion in next section) only preferred child changed over time.

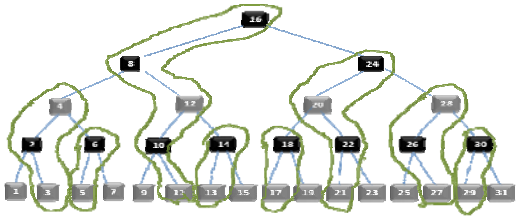
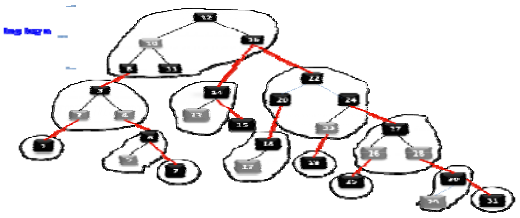


Figure 1. Balanced red-black with preferred paths using link-cut algorithm, total n/2 preferred paths.

We form a preferred path by joining all the preferred child chain from the root to its preferred child and to its preferred child and so on, until we reach a node which doesn't have a preferred child. Preferred path will contain at most O (logn) preferred child's which is height of P. We remove preferred from P, which makes the P into pieces of set of preferred paths. We generate a new balanced BST using these preferred paths which are called as Auxiliary tree has atmost (logn) nodes with height O(loglogn). Each node in Auxiliary tree we maintain its depth in P (used to maintain symmetric between reference tree and tango tree), min and max depth of the subtree over the nodes in auxiliary tree and isRoot bit that indicates its child is apart from the other auxiliary tree. We maintain the auxiliary tree as red-black tree which has a constant factor overhead [4]. The leaves of the auxiliary tree link to the root of the other immediate auxiliary tree. Longest preferred path auxiliary tree which contains a root of P will be top of the tango tree and will hang below with the in-order ordering.



Each preferred path is converted into auxiliary tree. Each auxiliary tree is hanged to leave of a top auxiliary tree satisfying BST property. Top auxiliary tree contains the root element of the reference tree P. Finally there n/2 auxiliary trees forms Tango tree T.

IV. TANGO SEARCH ALGORITHM

After construction tango tree T_0, reference trees P now there exists a Tango tree has the same set of data elements as reference tree P and each node in tango store depth with P. So, at any point of time we can construct reference tree by tango tree using the depth of the nodes which are part of P.

In this section, the behavior of a tango tree T_i, is going to be explained after access each x_i, how the structure of T_i of tango BST is going to change with P_i as stated above. To state this algorithm, first we are going to look the operations which are going to support by auxiliary tree.
The following operation should be performed by auxiliary tree:

- Searching for a node with its key value.
- Cutting an auxiliary tree in two
- Joining two auxiliary tree into one

All above operations should be performed O (logk) times where k is the no of nodes. We can claim that in red-black tree, we can perform SPLIT and CONCATENATE in O(logk) time[4,6] which are used in cutting and joining auxiliary tree. Auxiliary tree should maintain some addition information like depth, min depth, and max depth over each node in its subtree. Maintaining these auxiliary values doesn't affect the complexity of red-black tree.
Theorem 2: SPLIT (Ti, X): T is a red-black tree and x is a node in the tree, splitting the tree into two red black trees, where X is the root and left of X contains all nodes that are less than X and right of X contains all nodes greater than X.
Theorem 3: CONCATENATE (T1; T2; X): where T1 is a RB Tree whose nodes have key less than X, T2 is a Red-Black Tree whose nodes has key greater than x, merge the two trees in a single RB Tree, which contains all nodes in T1, T2 and a node with key = X.

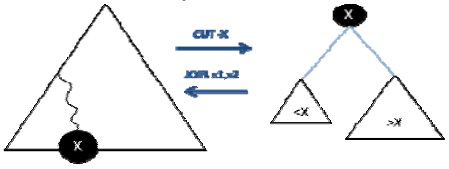


Figure 2. Red-Black Tree Split and Concatenate operations.

SPLIT and CONCATENATE in [1] is slightly differ from the standard functionality of these operations in terms of the number and type of input and output parameters. They are stated as *tango-cut* and *tango-join* operations The two operations describe above apply only to an auxiliary tree and do not cross into other auxiliary trees. We use the isRoot information for each node to avoid wandering in other trees.

To perform tango-cut on auxiliary tree A at depth d, we can cut the nodes whose Depth > d in a Red-Black Tree. Also, we can join two Red-Black trees into one which contains nodes with Depth > d, and we have also performed a cut to the other tree so that it's Depth > d nodes are all lost. The key observation here is in Red-Black Tree of any path, the keys of nodes that have Depth > d forms an interval [l; r] (because they are the intersection of a subtree of P and the path). We can get the nodes l and r by using the information in MinDepth and MaxDepth. Then we find the predecessor l_0 of l and the successor r_0 of r. All of these operations take O (log k) time in Red-Black Tree. To do cut, we do a SPLIT at l_0 and then SPLIT at r_0. Then we have a tree whose nodes has l_0< key < r_0 and is therefore all the nodes with Depth > d. We mark this tree has \hanged" and then do CONCATENATE at r_0 and l_0 respectively to finish cut operation

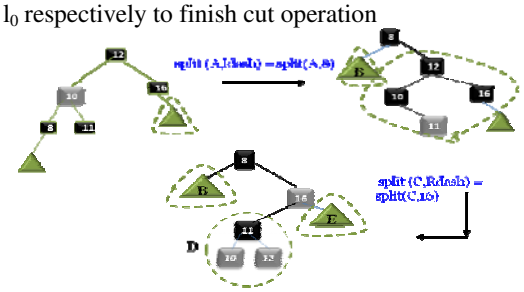


Figure 3. Tango-cut (Split) operation over auxiliary tree with 2 split operations.

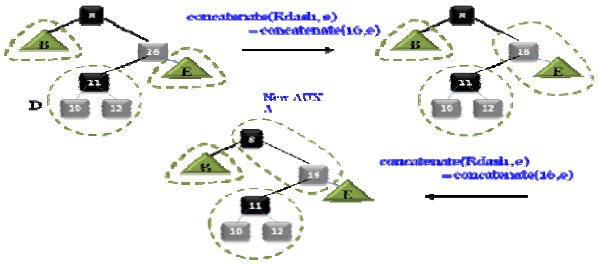


Figure 4. Tango-cut (concatenate) operation over the auxiliary tree after performing split and marking root bit with 2 concatenate operation.

To perform tango-join is similar to tango-cut. Suppose A is the tree with nodes Depth > d, B is the tree that do not have nodes with Depth > d. Observe that the key values in A must fall in between two adjacent keys l_0 and r_0 in B, we can do SPLIT at these two points, and then do two CONCATENATE's to join A and B.

To search node x, start from the root node of the top-auxiliary tree (root node of the reference tree P) we

traversal the tree in search of x, which traversals the way across other auxiliary trees. Say for example if we visit k auxiliary trees, each auxiliary tree takes O (loglogn) time to search. So we can say our entire search takes O (loglogn) time.

We need to update auxiliary tree whenever we touch the new auxiliary tree (preferred path changed in P). When the preferred path changed in P we need cut the path from any point and need to insert to another preferred path. Similarly we need to perform cut the auxiliary tree over any node, which takes 2 split operation, one marking root and one adding subtree operation by concatenate. And join the resultant auxiliary tree to new auxiliary tree. These split and concatenate will operate in O(loglogn) time. if k auxiliary trees are touched, no of changes to tango tree structure is O(loglogn)(k+1) times. Which infer as:

$$COST_{Tango}(K) <= O(loglogn) * IB(x)$$

So we can claim it remains O(loglogn)-competitive.

V. IMPLEMENTING DYNAMIZED TANGO TREE WHICH MAKING DATA STRUCTRE DYNAMIC

Slight modifications, can Make our data structure performing insert and delete pertaining all of the properties of Tango tree, including *O* (log log *n*) -competitiveness. In next coming sub sections we are going to describe how to make dynamized tango trees but first, we are going to discuss how much effort is required to perform insertion and deletion in reference tree P. There will be needed to maintain the invariants when nodes are inserted into and deleted from the reference tree so that the tree is balanced and every internal node has exactly one preferred child. To meet the balance requirements, considering rotations on the reference tree *P* (after insertion and deletion), and making *P* a dynamic red-black tree. The single preferred child requirement by making a constant number of switches prior to each rotation will be met. As the reference tree is implicitly maintained, so update operations over the reference tree must be simulated (e.g., rotations, pointer traversals) efficiently. In a Tango trees by simulating each of these operations turns out to cost *O* (loglogn) amortized time, so it is important that the corresponding reference tree will perform a sequence of m operations requires only *O* (*m*) virtual traversals and virtual rotations. (Finding the *location* of the update does *not* involve virtual traversals.) Red-Black trees require only *O* (1) amortized time to rebalance after an insert or delete [5], so we can say they meet this requirement.

A. Competitive analysis in Dynamic search trees

Before going to talk about the competitiveness of Dynamic Tango trees, we first define what to be meant for dynamic BST to be competitive. Consider a Dynamic binary search tree A which need to execute the access sequence X (x_1, x_2 , ... x_m) such as query(x_i), insert(x_i), and delete(x_i) to perform these operations cost of A is:

- To execute *the query* (xi), it costs touching each node on the path from the root to desired node xi.
- To execute *the insert* (xi), it must insert the node at a leaf and must pay for the traversal to get there. This is reasonable because *A* must search for xi to realize its BST does not contain xi.
- To execute *delete* (xi), it must pay for accessing xi and for performing rotations until xi has no children (at which time, the node can be removed).

A BST algorithm may perform any rotations it wishes at a cost of one per rotation during and after each operation. It costs simply the total number of nodes touched, plus the number of rotations. Without insert and delete operation, this definition would be identical to BST model.

B. Dynamic InterLeave Lower Bound

With our new definitions, we must prove a new lower bound for OPT (x_i). Our new lower bound is an extension of the one in [1], which is a variant of Wilber's 1st lower bound. As in the original principle of the interleave bound, for each node v in the initial reference tree P_0, we track if the last query in subtree_region (v) is in either L_v = left (subtree_region (v)) $\cup\{v\}$ or R_v = right (subtree_region (v)). Whenever the tracking for a node changes, we increment the dynamic interleave bound, DIB(T_i, x_i), by one. For an insert of v, we add the cost of querying pred(v) followed by succ(v) (because both of these nodes must be touched to insert v at a leaf). For a delete of v, we add the cost of querying pred(v), v, and succ(v) in succession because all three of these nodes must be touched in order to remove a node in Red-black tree. Whenever we rotate a node v, we reset the tracking of v and its parent to L_v but doesn't increase the interleave bound. Without insertions, deletions, and rotations, this definition would be identical to the original interleave bound.

C. Algorithm for insertion and deletion over Tango tree

To perform insertion in tango tree we need to consider a few things while performing this operation as the structure of the virtual reference tree P should maintain balance with maintaining the depth of the node in tango based on its parent node and it should be upgraded after a few sets of rotations for balancing.

The following steps will describe how the insert (x_i) works over tango:

1. Search (x_i) in tango tree T_i, while performing search using tango search algorithm to update the tango tree structure to maintain preferred path as P_i
2. If x_i is found, insertion stops as tree contains only unique data sets
3. Otherwise,
    - 3.1 Find predecessor and successor of parent (x_i)
    - 3.2 Perform tango-cut over the pred(x_i) and succ(x_i).
    - 3.3 To the resultant tree after tango-cut insert node to the parent (x_i) as a left / right child and mark the corresponding depth of the x_i as depth (parent (x_i)) +1

To perform deletion in tango tree, this algorithm is a bit complicated when compared to insertion, while performing delete (x_i) we need to consider several different cases: Node (x_i) with no child, node (x_i) with a single child and node (x_i) with two children. Here we are going to explain the case in which node contains 2 children the other cases are simpler to this one. After going to perform delete (x_i) we need to keep the modified tango tree T_i should be similar to virtual reference tree Pi. The following steps will describe how delete (x_i) works over tango:

1. Search (x_i) in tango tree T_i, while performing search using tango search algorithm to update the tango tree structure to maintain preferred path as P_i
2. If x_i is not found, deletion stops as the tree contains doesn't contain data set.
3. Otherwise,
    - 3.1 Find predecessor and successor of x_i these values are not the part of the auxiliary tree, they are found based on reference tree.
    - 3.2 First we swap x_i and pred(x_i) or succ(x_i) by performing tango-cut and tango-join over pred(x_i)
    - 3.3 The resultant will make x_i a new auxiliary tree will only one node with no children then simply delete the new auxiliary tree.
    - 3.4 While performing this sequence of operations make sure it should keep updating the depth of the nodes in the auxiliary tree.

However our Dynamic BST model doesn't support swap operations. This operation is implemented by rotating x_i to the leaf, removing xi and rotating pred(x_i) to take x_i's place. Because pred(x_i), xi, and succ(x_i) are located close together, O(1) rotations suffices. Then we change the field of the root of the Auxiliary tree to decrement the Depth and minDepth fields of every node in Auxiliary tree by one. These set of operations requiring only a constant number of field updates and reference pointer traversals, each costing O(log log n).

After the deletion, we might need to virtually rebalance the reference tree. In a Red-Black tree, we only need amortized O(1) pointer traversal and rotations for rebalancing.

D. Runtime analysis of insertion and deletion

For insertion and deletion to perform on a reference tree P we may need to rebalance the tree using rotation which takes only O (1) amortized, so the total amortized cost is O (loglogn) time.

For each insert operation, the number of tango cut and join operations, which each costs O(loglogn), performed during each insert query which is equal to increase in the dynamic interleave bound. So the cost can be the total number of dynamic interleave bound plus cost of rebalancing the new auxiliary tree after inserting the node which is in constant time of O (1) amortized.

For each delete operation, this is similar to an insert operation with 3 more extra querying operation of the query (pred (x_i)), query (x_i), succ (x_i) which each cost of O (loglogn) time for maintaining the tree structure balanced. Rebalancing the tree with x_i and its pred(x_i) or succ(x_i) and same time it should balance with reference tree P. The total cost of these extra querying and rebalancing of Red-Black tree for deleted node take O (loglogn) amortized time.

So by this runtime analysis we can say this upgraded tango tree is O(loglogn)-competitive.

VI. CONCLUSION

The main goal of this area of research is to extend the Tango tree T to allow insertion and deletion by satisfying its interleaving lower bound. We proved that insertion and deletion also apply on same bound where query case satisfies. Also showed that cost of rebalancing in Reference tree P after insertion and deletion is done in constant O(1) time as we use red-black tree for reference tree. The present results shows that upgraded tango tree satisfies all operation of online algorithms with total amortized cost of O(loglogn) time while satisfying O(loglogn)-competitiveness of tango tree.

As far as we can say based on our results this upgraded tango tree may be dynamically optimal? They are few open area of research in tango tree, weather this tree satisfies all necessary conditions of constant competitive BST? Like scanning Theorem, dynamic finger conjecture.

REFERENCES

[1] Erik D. Demaine, Dion Harmon, John Iacono, and Mihai P˘ atra¸ scu. "Dynamic Optimality–Almost". FOCS, 2004. Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pages 484–490. IEEE Computer Society, 2004.

[2] Robert Wilber. "Lower bounds for accessing binary search trees with rotations". SIAM Journal on Computing, 18(1):56–67, 1989.

[3] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. Journal of the ACM, 32(3):652–686, July 1985.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. "Introduction to Algorithms". MIT Press, second edition, 2001.

[5] R. E. Tarjan. "Sequential access in splay trees takes linear time". Combinatorica, 5(4):367–378, September 1985.

[6] R. Wein. "Efficient implementation of red-black trees with split and catenate operations". Technical report, Tel-Aviv University, 2005.

[7] Amr Elmasry. "On the sequential access theorem and deque conjecture for splay trees". Theoretical Computer Science, 314:459–466, 2004.

[8] Haripriyan Hampapuram and Michael L. Fredman. "Optimal biweighted binary trees and the complexity of maintaining partial sums". SIAM Journal on Computing, 28(1):1–9, 1998.

[9] Donald E. Knuth. "Optimum binary search trees". Acta Informatica, 1:14–25, 1971.

[10] Richard Cole, Bud Mishra, Jeanette Schmidt, and Alan Siegel. "On the dynamic finger conjecture for splay trees". Part I: Splay Sorting log n-Block Sequences. Siam J. Comput., 30:1–43, 2000.

[11] Richard Cole. "On the dynamic finger conjecture for splay trees". Part II: The Proof. Siam J. comput., 30:44–85, 2000.