**6.897: Advanced Data Structures** Spring 2007

# Lecture 3 — February 20, 2007

*Prof. Erik Demaine* *Scribe: Hui Tang*

## 1 Overview

In the last lecture we discussed Binary Search Trees and the many bounds which achieve $o(\lg n)$ per operation for natural classes of access sequences. This motivates us to search for a BST which is optimal (or close to optimal) on *any* sequence of operations.

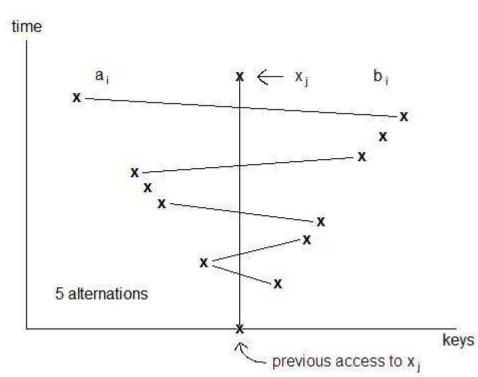
In this lecture we discuss lower bounds which hold for any Binary Search Tree (the Wilber bounds). Based on such a bound, we develop the Tango tree, which is an $O(\lg \lg n)$-competitive BST structure. The Tango upper bound and the Wilber lower bounds represent the tighest bounds on the offline optimal BST known to date.

## 2 Wilber Bounds

The two Wilber bounds are functions of the access sequence, and they are lower bounds for all BST structures. They imply, for instance, that there exists a *fixed* sequence of accesses which takes $\Omega(\lg n)$ per operations for *any* BST.

### 2.1 Wilber's 2nd Lower Bound

Let $\langle x_1, x_2, ..., x_m \rangle$ be the access sequence. For each access $x_j$ compute the following Wilber number. We look at where $x_j$ fits among $x_i, x_{i+1}, ..., x_{j-1}$ for all $i = j-1, j-2...$ that is, counting backwards from $j-1$ until the previous access to the key $x_j$. Now, we say that $a_i < x_j < b_i$, where $a_i$ and $b_i$ are the tightest bounds on $x_j$ discovered so far. Each time $i$ is decremented, either $a_i$ increases or $b_i$ decreases (more tightly fitting $x_j$), or nothing happens (the access is uninteresting for $x_j$). The Wilber number is the number of alternations between $a_i$ increasing and $b_i$ decreasing.



Wilber's 2nd lower bound [Wil89] states that the sum of the Wilber numbers of all $x_j$'s is a lower bound on the total cost of the access sequence, for any BST. It should be noted that this only holds in an amortized sense (for the total cost): the actual cost to access some $x_j$ may be lower than its Wilber number on some BSTs. We ommit the proof; it is similar to the proof of Wilber's 1st lower bound, which is given below.

An interesting open problem is whether Wilber's second lower bound is tight.

We now proceed to an interesting consequence of Wilber's 2nd lower bound: key-independent optimality. Consider a situation in which the keys are just unique identifiers, and, even though they are comparable, the order relation is not particularly meaningful. In that case, we "might as well" randomly permute the keys. In other words, we are interested in $E_\pi[OPT(\pi(x_1), \pi(x_2), ..., \pi(x_m)]$, where the expectation is taken over a random permutation $\pi$ of the set of keys.

Key-independent optimality is defined as usual with respect to the optimal offline BST.

**Theorem 1 (key independent optimality [Iac02]).** *A BST has the key-independent optimality property iff it has the working-set property.*

In particular, splay trees are key-independently optimal.

*Proof.* (sketch) We must show that:

$$E_\pi[\text{dynamic OPT}(\pi(x_1), \pi(x_2), ..., \pi(x_m)] = \text{Working-Set Bound } \Theta\left(\sum_{i=1}^{m} \lg t_i(x_i)\right)$$

The $O(\cdot)$ direction is easy: the working-set bound does not depend on the ordering, so it applies for any permutation. We must now show that a random permutation makes the problem as hard as the working-set problem:

- $wilber2(x_j)$ only considers the working-set of $x_j$
- define $W = \{\text{elements accessed since last access to } x_j\}$
- permutation $\pi$ changes $W$
- $x_j$ falls "roughly" in the middle with constant probability
- $E[\# \text{ times } a_i \text{ increases}] = \Theta(\lg |W|)$
- $E[\# \text{ times } b_i \text{ decreases}] = \Theta(\lg |W|)$
- **Claim:** We expect a constant fraction of the increases in $a_i$ and the decreases in $b_i$ to interleave $\Rightarrow E[wilber2(x_j)] = \Theta(\lg |W|)$

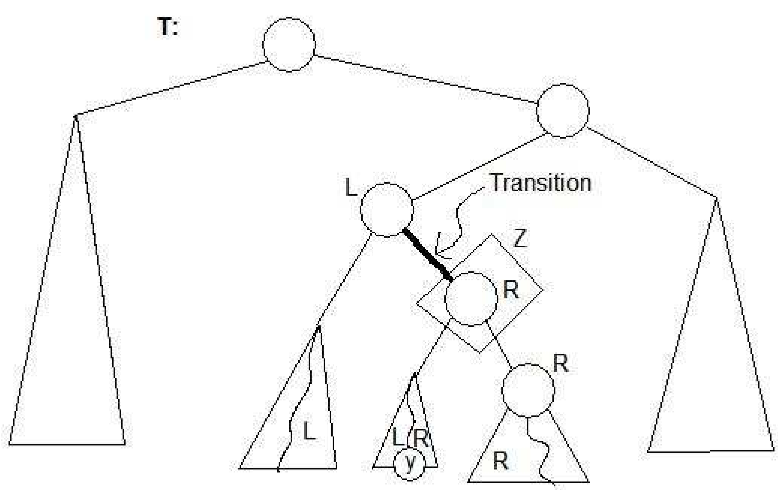
□

### 2.2 Wilber's 1st Lower Bound

Fix an arbitrary static lower bound tree $P$ with no relation to the actual BST $T$, but over the same keys. In the application that we consider, $P$ is a perfect binary tree. For each node $y$ in $P$, we label each access $x_i$ $L$ if key $x_i$ is in $y$'s left subtree in $P$, $R$ if key $x_i$ is in $y$'s right subtree in $P$, or leave it blank if otherwise. For each $y$, we count the number of interleaves (the number of alterations) between accesses to the left and right subtrees: $interleave(y) = \#$ of alternations $L \leftrightarrow R$.

Wilber's 1st lower bound [Wil89] states that the total number of interleaves is a lower bound for all BST data structures serving the access sequence $x$. It is important to note that the lower bound tree $P$ must remain static.

*Proof.* (sketch) We define the transition point of $y$ in $P$ to be the highest node $z$ in the BST $T$ such that the root-to-$z$ path in $T$ includes a node from the left and right subtrees of $y$ in $P$. Observe that the transition point is well defined, and does not change until we touch $z$. In addition, the transition point is unique for every node $y$.




We must touch the transition point of $y$ in the next access to a key labeled $L$ or $R$, that is, within the next 2 $interleave(y)$'s. This implies that the price we pay for accessing the sequence $x$ is greater than or equal to half the number of interleaves, that is, $pay \geq \frac{interleave(x)}{2}$.

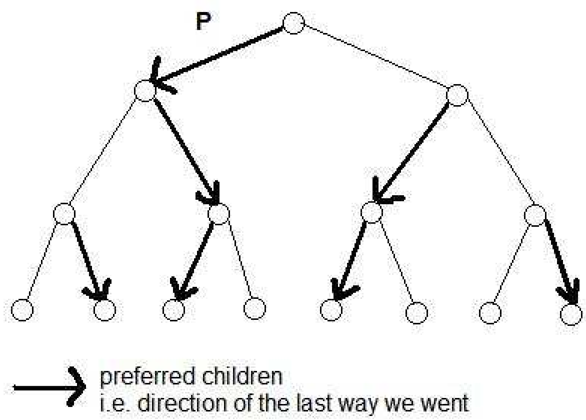
□

# 3 Tango Trees

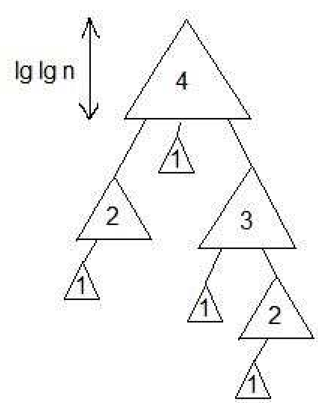
Tango trees [DHIP04] are an $O(\lg \lg n)$-competitive BST. They represent an important step forward from the previous competitive ratio of $O(\lg n)$, which is achieved by standard balanced trees. The running time of Tango trees is $O(\lg \lg n)$ higher than Wilber's first bound, so we also obtain a bound on how close Wilber is to $OPT$. It is easy to see that if the lower bound tree is fixed without knowing the sequence (as any online algorithm must do), Wilber's first bound can be $\Omega(\lg \lg n)$ away from $OPT$, so one cannot achieve a better bound using this technique.

To achieve this improved performance, we divide a BST up into smaller auxiliary trees, which are balanced trees of size $O(\lg n)$. If we must operate on $k$ auxiliary trees, we can achieve $O(k \lg\lg n)$ time. We will achieve $k = 1+$ the increase in the Wilber bound given by the current access, from which the competitiveness follows.

Let us again take a perfect binary tree $P$ and select a node $y$ in $P$. We define the preferred child of $y$ to be the root of the subtree with the most recent access (i.e. the preferred child is the left one iff the last access under $y$ was to the left subtree). If $y$ has no children or its children have not been accessed, it has no preferred child. An interleave is equivalent to changing the preferred child of a node, which means that the Wilber bound is the number of changes to preferred children.




Now we define a preferred path as a chain of preferred child pointers. We store each preferred path from $P$ in a balanced auxiliary tree that is conceptually separate from $T$, such that the leaves link to the roots of the auxiliary trees of "children" paths.




Because the height of $P$ is $\lg n$, each auxiliary tree will store $\leq \lg n$ nodes. A search on an auxiliary tree will therefore take $O(\lg\lg n)$ time, so the search cost for $k$ auxiliary trees $= O(k \lg\lg n)$.

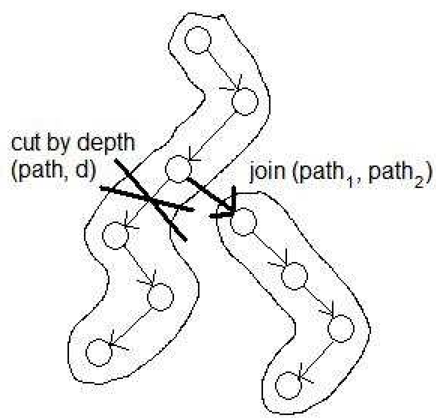
A preferred path is not stored by depth (that would be impossible in the BST model), but in the sorted order of the keys.

### 3.1 Searching Tango trees

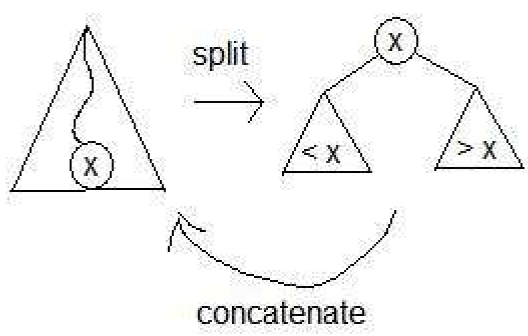
To search this data structure for node $x$, we start at the root node of the topmost auxiliary tree (which contains the root of $P$). We then traverse the tree looking for $x$. It is likely that we will jump between several auxiliary trees – say we visit $k$ trees. We search each auxiliary tree in $O(\lg \lg n)$ time, meaning our entire search takes place in $O(k \lg \lg n)$ time. This assumes that we can update our data structure as fast as we can search, because we will be forced to change $k - 1$ preferred children (except for startup costs if a node has no preferred children).

### 3.2 Balancing Auxiliary Trees

The auxiliary trees must be updated whenever preferred paths change. When a preferred path changes, we must cut the path from a certain point down, and insert another preferred path there.




We note that cutting and joining resemble the *split* and *concatenate* operations in balanced BST's. However, because auxiliary trees are ordered by key rather than depth, the operations are slightly more complex than the typical *split* and *concatenate*.




Luckily, we note that a *depth* $> d$ corresponds to an interval of keys. Thus, cutting from a point down becomes equivalent to cutting a segment in the sorted order of the keys. In this way, we can change preferred paths by cutting a subtree out of an auxiliary tree using two split operations and adding a subtree using a concatenate operation. To perform these cut and join operations, we label the roots of the path subtrees with a special color and pretend it's not there. This means we only need to worry about doing the *split* and *concatenate* on a subtree of height $\lg n$ rather than trying to figure out what to do with all the auxiliary subtrees hanging off the path subtree we are interested in. We know that balanced BSTs can support *split* and *concatenate* in $O(\lg size)$ time, meaning they all operate in $O(\lg \lg n)$ time. Thus, we remain $O(\lg \lg n)$-competitive.

## References


[DHIP04] Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Patrascu. Dynamic optimality — almost. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, pages 484–490. IEEE Computer Society, 2004.

[Iac02] John Iacono. Key independent optimality. In *ISAAC '02: Proceedings of the 13th International Symposium on Algorithms and Computation*, pages 25–31. Springer-Verlag, 2002.

[Wil89] R. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM J. Comput.*, 18(1):56–67, 1989.