6.897: Advanced Data Structures Spring 2005

Lecture 4 — February 10, 2005

Prof. Erik Demaine *Scribe: Mike Ebersol*

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 counting backwards from $j-1$ until the previous access to 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. Specifically, we will assume that the order relation is just random. In other words, we are interested in $E[OPT(b(x_1), b(x_2), ..., b(x_n)]$, where the expectation is taken over a uniformly random bijection b on the set of keys.

Key-independent optimality is defined as usual with respect to the optimal offline BST. Note that the offline optimum knows the bijection (alternatively, we take the optimum for each bijection).

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) Take a uniformly random bijection b. We must show that:

$$E[OPT(b(x_1), b(x_2), ..., b(x_n)] = \Theta(\sum \lg t_i(x_i))$$

The $O(\cdot)$ direction is easy: the working-set bound does not depend on the ordering, so it applies for any bijection. We must now show that a random bijection makes the problem as hard as the working-set problem. We do that by showing that the Wilber number of any $b(x_j)$ is in expectatation $\lg t_j(x_j)$. Fix j. For each $x_i, i < j$, we remove any previous accesses to x_i; we also ignore anything prior to the last access to x_j. We are now left with $t_j(x_j)$ elements. A random b induces a random permutation of this working set. The following are classic results in probabilistic analysis:

- x_j falls "roughly" in the middle with constant probability
- $E[\text{changes to } a_i] = \Theta(\lg t_j(x_j))$
- $E[\text{changes to } b_i] = \Theta(\lg t_j(x_j))$
- the changes to a_i and the changes to b_i interleave in $\Omega(\lg t_j(x_j))$ positions, so $E[\text{Wilber number}] = \Theta(\lg t_j(x_j)) = \text{working-set bound}$

□

2.2 Wilber's 1st Lower Bound

Fix an arbitrary static lower bound tree P with not 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, and every access x_j, we see whether the access is in the left subtree of y, the right subtree of y, or neither. For each y, we count the number of interleaves (the number of alterations) between accesses to the left subtree and to the right subtree.

Wilber's 1st lower bound [Wil89] states that the total number of interleaves minus n 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.

We now sketch the proof of this lower bound. Let us set up an amortization argument. 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. Also, it is unique for every node y.

Each time an interleave through y occurs, we drop a marble on the transition point of y. Each time the BST algorithm touches a node, we collect the marble that may be present in that node. We now argue that a node never contains more than one marble. Before another interleave through y occurs, we must touch the transition point of y. Thus, we never deposit another marble without having picked up the previous one. Then, the number of interleaves is a lower bound on the number of marbles deposited. The number of marbles picked up is a lower bound on the total cost. At the end, there are at most one marble left in any node, so at most n marbles in total. Then, the cost is at least the total number of interleaves minus n.

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 is 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 nodes, where each node is followed by its preferred child. We store each preferred path from P in an auxiliary tree in the actual BST. 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.

We now link together the auxiliary trees to complete the data structure. The leaves in an auxiliary tree will have as children the roots of the auxiliary trees corresponding to preferred paths branching off the current preferred path. It should also be noted that 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. Note that cutting from a point down is actually equivalent to cutting a segment in the sorted order of the keys. Thus, we change preferred paths by cutting a subtree out of an auxiliary tree using two split operations and adding a subtree using a concatenate operation. We know that balanced BSTs can support split and concatenated 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.