6.897: Advanced Data Structures Spring 2010

Lecture 2 — February 4, 2010

Prof. Erik Demaine

1 Overview

In the last lecture we discussed Binary Search Trees(BST) and introduced them as a model of computation. A quick recap: A search is conducted with a pointer starting at the root, which is free to move about the tree and perform rotations; however, the pointer must at some point in the operation visit the item being searched. The cost of the search is simply the total number of distinct nodes in the trees that have been visited by the pointer during the operation. We measure the total cost of executing a sequence of searches $S = \langle s_1, s_2, s_3 \ldots \rangle$, where each search s_i is chosen from among the fixed set of n keys in the BST.

We have witnessed that there are access sequences which require $o(\log(n))$ time per operation. There are also some deterministic sequences on n queries (for example, the bit reversal permutation) which require a total running time of $\Omega(n \log(n))$ for any BST algorithm. This disparity however does not rule out the possibility of having an instance optimal BST. By this we mean: Let $OPT(S)$ denote the minimal cost for executing the access sequence S in the BST model, or the cost of the best BST algorithm which has access to the sequence apriori. It is believed that splay trees are the "best BST". However, they are not known to have $o(\log(n))$ competitive ratio. Also, notice that we are only concerned with the cost of the specified operations on the BST and we are not accounting for the work done outside the model, say, the computation done for rotations etc.

This motivates us to search for a BST which is optimal (or close to optimal) on *any* sequence of search. Given splay trees satisfy a number of properties like static optimatlity, working set bound, dynamic-finger bound and linear traversal; they are a natural candidate for the dynamic optimality. They are notoriously hard to analyse and understand and sometimes appear magical.

So, this led researchers to look for alternative approaches to build a dynamically optimal BST. The best guarantee so far is the $O(\log\log(n))$ competitive ratio achieved by the Tango Trees - we shall see them in the later part of the lecture.

Another perspective, is the recently proposed geometric view of the BST [DHIKP09]. In this approach, an correspondence between the BST model of computation and points in $\mathbb{R}^2$ is given. Informally, call a set P of points arborally satisfied if, for any two points $a, b \in P$ not on a common horizontal or vertical line, there is al teast one point $P \setminus \{a, b\}$ in the axis parallel rectangle defined by a and b. Each search is mapped into the$\mathbb{R}^2$ in the following way: $P = \{(s_1, 1), (s_2, 2) \ldots (s_n, n)\}$. In this lecture, we show prove the following : Finding the best BST for an access sequence S is equivalent to finding the minimal cardinality superset $P' \supseteq P$ that is arborally satisfied.

Another candidate for dynamic optimality is the *Greedy* BST algorithm; which was originally proposed for the offline case. Informally, the *Greedy* BST does the following: when searching for an item, it re-arranges the nides on the search path to minimize the cost of the future searches, by making necessary rotations. All this is being with a hunch that going off the search path should not help by much. It was conjectured that this algorithm is a constant factor approximation for

$OPT(S)$.

We then give lower bounds on $OPT(S)$. So far, only two lower bounds developed by Wilber were known. Wilber's first bound is used extensively for to show that Tango trees are $o(\log\log(n))$ competitive. Wilber's second bound is stronger and folklore has it is indeed tight. The state-of-the-art is, however, the upper bound of Tango trees and the Wilber lower bounds represent the tighest bounds on the offline optimal BST.

2 BST and Arborally Satisfied Sets

We shall now define what constitutes as an BST execution. We use the letters n and m to refer to the size of a BST, and the total number of search operations performed. Without loss of generality, we assume that the key space is $1, 2, \dots n$. We now define an execution of a BST algorithm.

Definition 1 (BST Execution)**.** *Given a search sequence $S = \langle s_1, s_2 \dots s_m \rangle$, we say a BST algorithm executes S by an execution $E = \langle T_0, \tau_1 \rightarrow \tau_1\prime \dots, \tau_m \rightarrow \tau_m\prime \rangle$, where $\tau_i \rightarrow \tau_i\prime$ is the reconfiguration the tree has undergone while searching for the key s_i and $s_i \in \tau_i$ for all i.*

We shall use the notation $\Box ab$ to denote a axis aligned rectangle with points a and b as its corners. We assume that the x-coordinates of all the points are unique.

Definition 2 (Arborally Satified Set (ASS))**.** *A pair of points (a, b) (or their induced rectangle $\Box ab$) is arborally satisfied to a point set P if a and b are orthogonally collinear(horizontally or vertically aligned) or if there is at least one point from the $P \setminus \{a, b\}$ in $\Box ab$. A point set P is arborally satisfied if all the pairs of points in P are arborally satisfied with respect to P.*

Definition 3 (Geometric View of BST (GBST))**.** *The geometric view of a BST execution E is the point set $P(E) = \{(x, y) | x$ is visited in searching for item $y\}$.*

Lemma 4 (GBST $\Rightarrow$ ASS)**.** *The point set $P(E)$ for any BST execution is arborally satisfied.*

Proof Sketch: Assume for contradiction that we can find two points (a, i) and (b, j) and yet no other nodes were touched in the closed time interval $[i, j]$. Let c be the lowest common ancestor of a and b in the BST after i^th search. We have two cases:

$c \neq a$ Then c must be touched at time i to get to a in the i^{th} search and c must have a key value between a and b. Contradiction.

$c = a$ Then, at i, a is an ancestor of b is not touched from i to j. Thus a will remain on the access path of b and that means a must be an ancestor of b at j and will be touched while accessing b at j. Contradiction. $\Box$

Lemma 5 (ASS $\Rightarrow$ BST)**.** *For any arborally satisfied point set X, there exists a BST execution E with $P(E) = X$. We call E the arboral view of X and write $P^{-1}(X) = E$.*

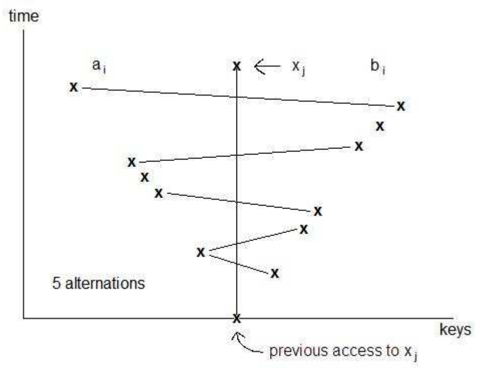
We let minASS(S) be the size of the smallest arborally satisfied super set of $P(S)$, we have OPT(S) = minASS($P(S)$).

3 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.

3.1 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 π of the set of keys.

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

Theorem 6 (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 π 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 $\Rightarrow E[wilber2(x_j)] = \Theta(\lg |W|)$

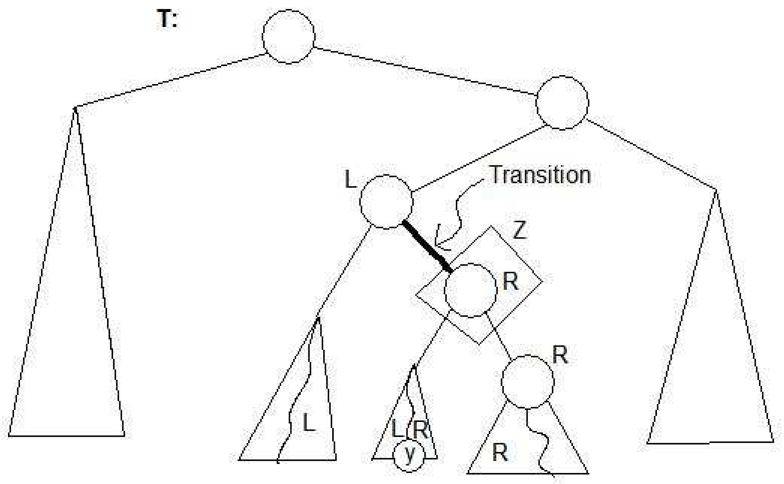
□

3.2 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, $pay \geq \frac{interleave(x)}{2}$.

$\square$

4 Independent Rectangle Bounds

Definition 7. *A pair of rectangles $\square ab$ and $\square cd$ are called independent if the rectangles are not arborally satisfied and no corner of either rectangle is strictly inside each other.*

Claim 8. *If a point set Y contains an independent set of rectangles, then the minASS(Y) $\geq \frac{|I|}{2} + |Y|$. In particular, if $Y = P(S)$ for an access sequence S, then $OPT(S) \geq \frac{|I|}{2}$.*

Definition 9 ($\boxslash$-Rectangles)**.** *We call a rectangle $\square ab$ to be $\boxslash$-rectangle ($\boxbslash$-rectangle) if the slope of the line segment $\overline{ab}$ is positive (negative).*

From now on, all lemmas and theorems will be stated using $\boxslash$-rectangles; but they can symetrically applied to $\boxbslash$-rectangles as well.

A point set is $\boxslash$ satisfied if every pair of points (a, b) that form a $\boxslash$-rectangle $\square ab$ si arborally satisfied. In other words, $\boxbslash$-rectangles need not be satisfied for $\boxslash$ satisfication. Let $\text{minASS}_{\boxslash}(X)$ be the size of the minimum $\boxslash$-satisfied superset of X, which is nothing more than a GreedyAss that ignores $\boxbslash$-rectangles:

The algorithm GreedyAssis as follows: Sweep a line along the increasing y coordinate. When considering a point on this line, for each unsatisfied $\boxslash$-rectangle formed by this point and another one (below the line), add the rectangle's northwest corner on the line to make it satisfied. Let $\text{add}_{\boxslash}(X)$ be the final set of added points.

We now use $\text{minASS}_{\boxslash}(X)$ to independent rectangle bounds. We show that lower bound output by this algorithm is at least $\frac{1}{4}maxIRB(X) + \frac{1}{2}|X|$, and is within a constant factor of the best independent rectangle bound.

Theorem 10. *If X contains an independent set I of rectangles, then $minASS_{\boxtimes}(X) \geq \frac{|I|}{2} + |X|$.*

Proof: We prove this using a chain of deductions:

$$\begin{aligned}
& \frac{1}{2}\max\{\text{Wilber I}(X), \text{Wilber II}(X)\} + |X| \\
\leq\ & \frac{1}{2}\text{maxIRB}(X) + |X| \\
\leq\ & \text{minASS}_{\boxtimes}(X) \\
\leq\ & \text{minASS}_{\boxslash}(X) + \text{minASS}_{\boxbslash}(X) \\
\leq\ & |\text{add}_{\boxslash}(X)| + |\text{add}_{\boxbslash}(X) + 2|X| \\
=\ & |\text{IRB}_{\boxslash}(X)| + |\text{IRB}_{\boxbslash}(X) + 2|X| \\
\leq\ & 2\text{maxIRB}(X) + 2|X| \\
\leq\ & 2\text{maxIRB}(X) + 4|X| \\
\leq\ & 4\text{minASS}_{\boxtimes}(X) \\
\leq\ & 4\text{minASS}(X)
\end{aligned}$$

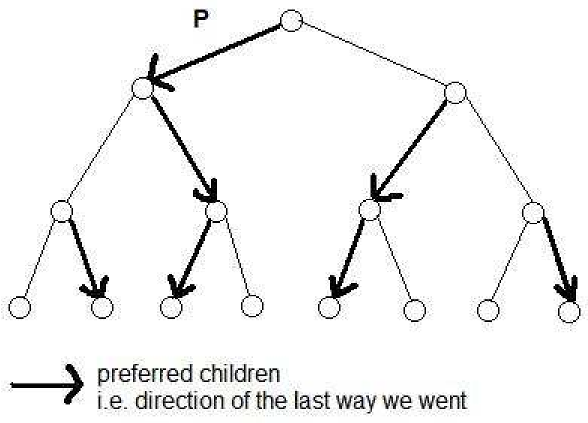
□

5 Tango Trees

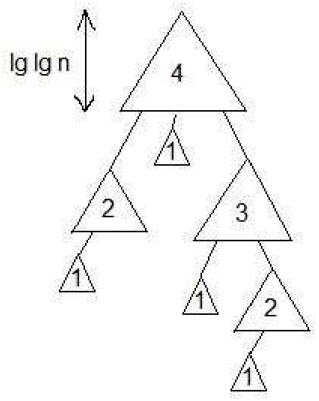
Tango trees [DHIP04] are an $O(\lg\lg n)$-competitive BST. They represent an important step forward from the previous competitive ratio of $O(\lg n)$, which is achieved by standard balanced trees. The running time of Tango trees is $O(\lg\lg n)$ higher than Wilber's first bound, so we also obtain a bound 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 $= O(k \lg \lg n)$.

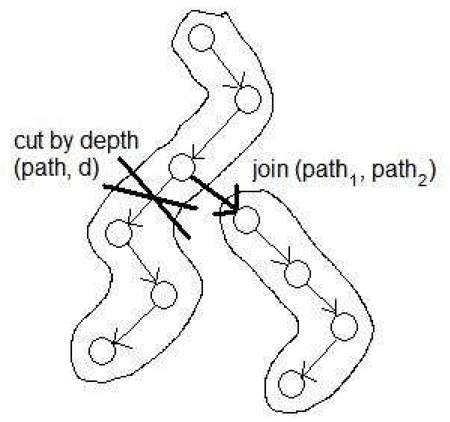
A preferred path is not stored by depth (that would be impossible in the BST model), but in the sorted order of the keys.

5.1 Searching Tango trees

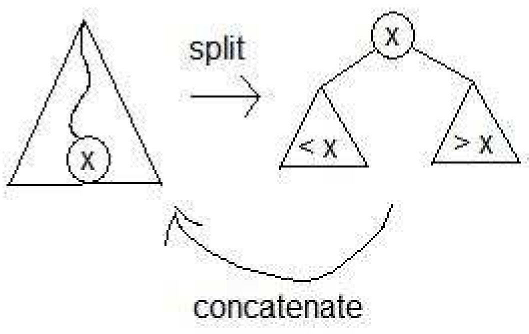
To search this data structure for node x, we start at the root node of the topmost auxiliary tree (which contains the root of P). We then traverse the tree looking for x. It is likely that we will jump between several auxiliary trees – say we visit k trees. We search each auxiliary tree in $O(\lg \lg n)$ 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).

5.2 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.

[DHIKP09] , Demaine, Erik D. and Harmon, Dion and Iacono, John and Kane, Daniel and Pătraşcu, Mihai. The geometry of binary search trees. *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms. 2009.* 496–505. New York, New York. Society for Industrial and Applied Mathematics.