6.851: Advanced Data Structures Spring 2012

Lecture 6 — March 6, 2012

Prof. Erik Demaine *Scribes: Aakanksha Sarda (2012), David Field (2012), Leonardo Urbina (2012), Prasant Gopal (2010), Hui Tang (2007), Mike Ebersol (2005)*

1 Overview

In the last lecture we discussed representing executions of binary search trees as point sets. A BST execution is represented as the point set of all nodes touched during the execution of the algorithm. A point set represents a valid BST execution if and only if it is arborally satistified. A point set is arborally satisfied iff any rectange spanned by two points not on a horizonal or vertical line contains another point. This point set representation of binary searches suggested an offline greedy algorithm for generating valid point sets from a set of queries, by adding the points required to make the point set arborally satisfied row by row. This greedy algorithm is conjectured to be O(optimal). The offline greedy algorithm can be simulated online.

In this lecture we are going to focus on lower bounds. We will be using the method of independent rectangles to establish lower bounds. We will discuss three applications of the method of independent rectanges Wilber 1, Wilber 2, and signed greedy. The signed greedy algorithm is almost the same as greedy, but it is a lower bound. It is conjectured, but not proven that these two algorithms are within a constant factor of each other. We will also discuss tango trees which are $O(\lg \lg n)$-competitive.

2 Independent Rectangle Bounds

Now we establish lower bounds on the performance of binary search trees using the method of independent rectangles. A pair of rectangles are called independent if the rectangles are not arborally satisfied and no corner of either rectangle is strictly inside the other rectangle. The Independent Rectangle bounds state that the number of additional points required to make a point set arborally satisfied is greater than or equal to the maximum number of independent rectangles in that point set. Due to the equivalence between arborally satisfied point sets and binary search trees, this puts a lower bound on the performance of binary search trees.

Given any two points a and b, there is a unique rectangle with a as one of its vertices, and b as the opposite vertex. Let us refer to this rectangle as rectangle ab. If the line $\overline{ab}$ has positive slope, let us call this rectangle a S-rectangle. S stands for the slash, '/', formed by the line. If the line $\overline{ab}$ has negative slope, let us call this rectangle a B-rectangle. B stands for the backslash, '\', formed by the line.

Definition 1. *A pair of rectangles ab and cd are called independent if the rectangles are not arborally satisfied and no corner of either rectangle is strictly inside of the other rectangle.*

Definition 2. *Given a point set P, let OPT be the cardinality of the smallest arborally satisfied superset of P. (OPT may also be used to refer to the minimal superset)*

Definition 3. *Given a point set P, let $MAXIND_{\square}$ be the cardinality of the largest set of independent rectangles that is a subset of the rectangles defined by P. Let $MAXIND_{\boxslash}$ and $MAXIND_{\boxbslash}$ be defined in the same way but for independent S-rectangles and independent B-rectangles, respectively.*

Theorem 4. *Given a point set S, $OPT \geq |input| + 1/2\, MAXIND_{\square}$*

From now on, all lemmas, definitions, and theorems will be stated using S-rectangles; but they can symetrically applied to B-rectangles as well.

Definition 5 (S-satisfied)**.** *We call a point set S-satisfied if all S-rectangles in it have another point in them.*

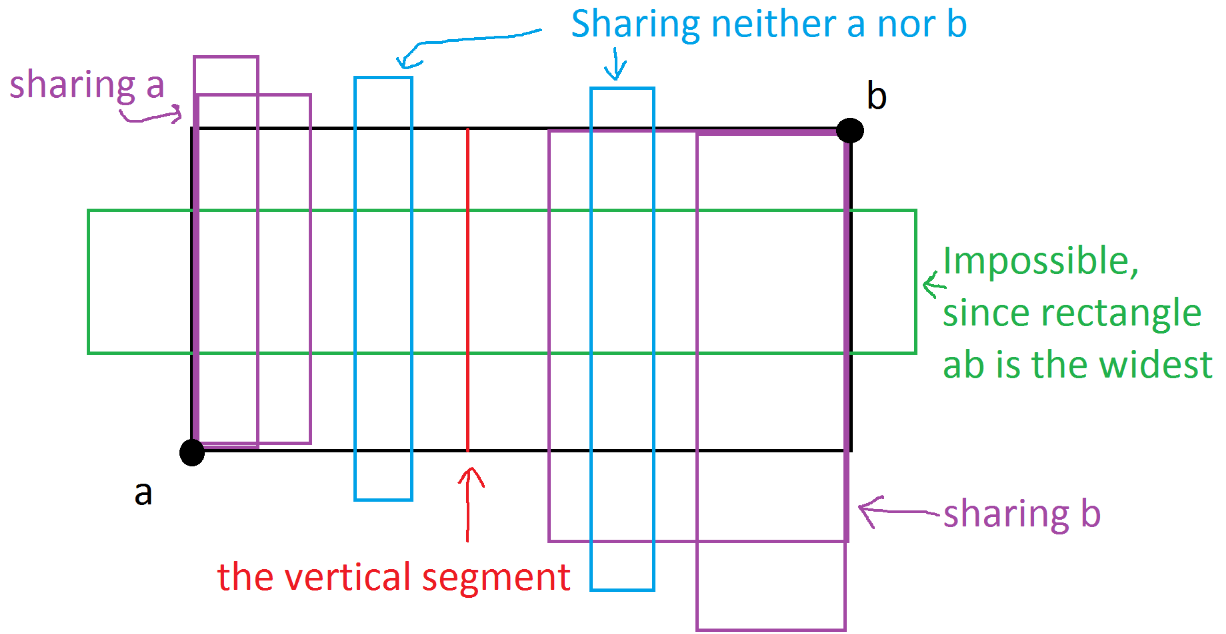
Definition 6 ($OPT_{\boxslash}$)**.** *Given a point set P, $OPT_{\boxslash}$=the cardinality of the smallest S-satisfied superset of P.*

Lemma 7. $OPT_{\boxslash} \geq |input| + MAXIND_{\boxslash}$

2.1 Proof of Lemma

Proof. Assume the points in the input point set P have distinct x and y. If this is not the case, the input can be skewed so that the input x and y are all distinct. Consider the largest set of independent S-rectangles that is a subset of the S-rectangles defined by P. (or any one of the largest sets if it is not unique) We will show by a charging argument that $OPT_{\boxslash} \geq |\text{input}| + MAXIND_{\boxslash}$.

Existance of vertical segment: Consider the widest rectangle in the independent set, shown in the figure below. Call the point at its lower left vertex a and the point at its upper right vertex b. There must exist a vertical segment going from the botton edge to the top of edge of this rectangle which has no points in other rectangles. This is because all rectangles who's interiors intersect the widest rectangle's interior must share vertex a with the widest rectangle, share vertex b with the widest rectangle, or share neither. The sharing a rectangles must be left of the sharing b rectangles by independence (they cannot just share edges due to the distinct x coordinates). The sharing neithers cannot have one vertical edge intersecting a particular a or b sharing rectangle, and one not, because the rectangles must be independent. Since the a and b sharing rectangles cannot intersect, and the sharing neither rectangles must fit in between the sharing b and sharing a rectangles, there will be room for a vertical segment v.



Selection of p and q: Take p to be the topmost rightmost point from $OPT_{\boxslash}$ in S-rectangle ab to the left of the vertical line segment. Let q be the bottommost leftmost point from $OPT_{\boxslash}$ in S-rectangle ab to the right of the line segment and not below p.
Assume these points are not horizontal. If there was any point from $OPT_{\boxslash}$ in S-rectangle pq then that point would either to the left of the vertical segment and farther right and up than p or to the right of the vertical segment and father down and right than q, which cannot be the case by the definition of p and q. Then S-rectangle pq is not satisfied, which is a contradiction. Thus segment pq is horizontal.

Charging: Remove the S-rectangle ab from the set of independent S-rectangles and charge it to the horizontal line containing p and q. (Note that a and b are not being removed, just the S-rectange ab). The pair of points p and q will not be charged together again since the vertical segment that intersects the horizontal line between them in not intersected or contained by any other rectangles, so no other rectangle contains segment pq. Any set of n charges to a horizonal line must charge $n + 1$ distinct points, since no pair of points can be charged twice, and all pairs of points charged must be consequtive points on the line. At most 1 of these points can be from the input. Thus the number of added points on a horizontal line is greater than or equal to the number of charges to that line. Thus $OPT_{\boxslash} \geq |\text{input}| + MAXIND_{\boxslash}$ as desired. $\square$

2.2 Proof of Theorem

Proof. Using the lemma:

$$\begin{aligned} OPT &\geq \max(OPT_{\boxslash}, OPT_{\boxbslash}) && (1)\\ &\geq 1/2(OPT_{\boxslash} + OPT_{\boxbslash}) && (2)\\ &\geq |input| + 1/2\, MAXIND_{\boxslash} + 1/2\, MAXIND_{\boxbslash} && (3)\\ &\geq input| + 1/2\, MAXIND_{\square} && (4) \end{aligned}$$

□

3 Lower Bounds

3.1 Wilber's second lower bound [1]:

Wilber's second bound can be introduced as follows. Given the input (access) point set:

1. For each point p:
   (a) Look at all of the orthogonally visible points below p
   (b) Count the number of alternations between left/right of p
2. Sum over all p

Proof: Add an independent rectangle of different sign for each alternation.

OPEN: OPT = ΘWilber2

3.2 Key-independent Optimality [2]:

Suppose now that that the key values are "meaningless". In this case we could just permute them randomly. With this mind we can make the following claim:

$$\mathbf{E}[\text{OPT}] = \text{working-set bound}$$

Hence, splay trees are key-independent optimal.

Sketch of Proof: The expected number of changes to to max in random permutation is

$$\mathbf{E}[\text{Wilber2}(x_i)] = \Theta(\log t_i)$$

Wilber's First Lower Bound [1]: We can arrive at Wilber's first lower bound as follows. Fix a lower-bound tree P having the input as its keys. Then:

1. For each node y of P: Count the number of alternations in the access sequence $x_1, x_2, \ldots, x_n$ between accesses in left and right subtrees of y (ignoring accesses to y or outside of y's subtree.
2. Sum over all y

Proof: Add an independent rectangle for each alternation.

Example: bit-reversal sequence Consider the sequence of numbers that results from the list of all non-negative integers, and reading their binary representations backwards. Namely, start with

$$0, 1, 2, 3, 4, 5, 6, 7, \ldots$$

Which written in binary is,

$$000_1, 001_2, 010_2, 011_2, 100_2, 101_2, 110_2, 111_2$$

Reversing these yields:

$$000_1, 100_2, 010_2, 110_2, 001_2, 101_2, 011_2, 111_2$$

In other words

$$0, 4, 2, 6, 1, 5, 3, 7$$

If we use this as the access sequence in a perfect binary tree we see that the number of alternations at y equals the size of the subtree rooted at y. From this it follows that:

$$\begin{aligned} & \text{Wilber1} = \Theta(n \log n) \\ \Rightarrow \quad & \text{OPT} = \Theta(n \log n) \end{aligned}$$

OPEN: Is it true that for every access sequence there exist a tree P such that

$$\text{OPT} = \Theta(\text{Wilber1})$$

4 Tango Trees

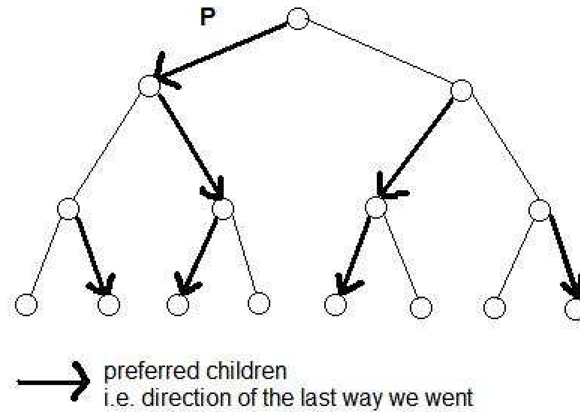
Tango trees [DHIP04] are an $O(\lg \lg n)$-competitive BST. They represent an important step forward from the previous competitive ratio of $O(\lg n)$, which is achieved by standard balanced trees. The running time of Tango trees is $O(\lg \lg n)$ higher than Wilber's first bound, so we also obtain a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

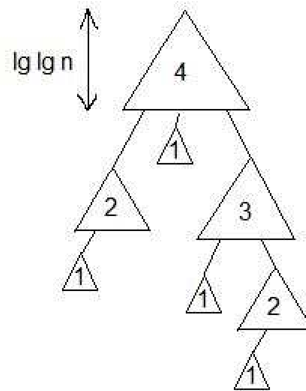
Let us again take a perfect binary tree P and select a node y in P. We define the preferred child of y to be the root of the subtree with the most recent access i.e. the preferred child is the left one

iff the last access under y was to the left subtree. As in the figure below, each node has at most one preferred child, and "preferred paths" through the tree are formed.

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 total number of times a preferred child was changed.



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.

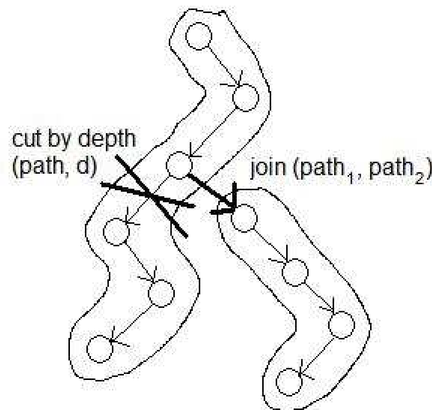
4.1 Searching Tango trees

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

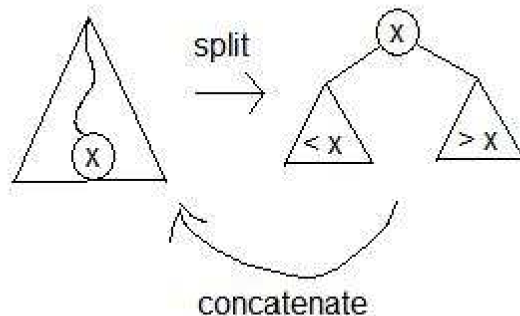
between several auxiliary trees – say we visit k trees. We search each auxiliary tree in $O(\lg \lg n)$ time, meaning our entire search takes place in $O(k \lg \lg n)$ time. This assumes that we can update our data structure as fast as we can search, because we will be forced to change $k-1$ preferred children (except for startup costs if a node has no preferred children).

4.2 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 - from min(path below depth d) to max (path below depth d). 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.

5 Signed greedy algorithm

This section will build on the Greedy algorithm from last lecture, to present a modified "Signed Greedy" algorithm whose run-time serves as a lower bound.

5.1 Algorithm

Conduct the greedy-algorithm sweep as before, considering one point row at a time. However, for signed-greedy, we consider only unsatisfied $\boxslash$-rectangles (or only $\boxbslash$-rectangles) when deciding whether to add points.

Thus, only the $\boxslash$-rectangles (or only the $\boxbslash$-rectangles) will be satisfied. Also, similarly to the original greedy algorithm, every added point will correspond to an independent $\boxslash$-rectangle.

Definition 8. *Define $OPT_{\boxtimes}$ to be the smallest union of the $\boxslash$-satisfying and $\boxbslash$-satisfying supersets of the points.*

5.2 Signed Greedy provides a lower bound

Theorem 9. $\max\{\boxslash\text{-}greedy, \boxbslash\text{-}greedy\} = \theta(biggest\ independent\text{-}rectangle\ LB)$

Proof.

$$
\begin{aligned}
\mathrm{OPT}_{\boxtimes} &\geq |input| + \frac{1}{2}\max(\text{number of independent rectangles}) &(5)\\
&\geq \frac{1}{2}\max(\boxslash\text{-greedy}, \boxbslash\text{-greedy}) &(6)\\
&\geq \frac{1}{2}\max(\mathrm{OPT}_{\boxslash}, \mathrm{OPT}_{\boxbslash}) &(7)\\
&\geq \frac{1}{4}(\mathrm{OPT}_{\boxslash} + \mathrm{OPT}_{\boxbslash}) &(8)\\
&\geq \frac{1}{4}\mathrm{OPT}_{\boxtimes} &(9)
\end{aligned}
$$

□

Thus, a constant factor 'sandwich' is created, and all the expressions (1) to (5) must be within a constant factor of each other. In particular, (1) - which is the independent rectangle lower bound - and (2) must be within a constant factor of each other, and the theorem follows.

5.3 Greedy vs. ⧄-greedy

Greedy corresponds to a valid BST, and provides and upper bound on the run-time of OPT. Signed-greedy ($\mathrm{OPT}_{\boxtimes}$) is not a valid BST, but it provides a lower bound on the run-time of every valid BST. The key difference between the point-sets obtained from Greedy and from Signed-Greedy is that Signed-Greedy fails to take into account the interactions between the ⧄-rectangles and the ⧅-rectangles, i.e. the points added to satisfy ⧄-rectangles may introduce new unsatisfied ⧅-rectangles or vice versa.

Project idea: Compare upper-bound and lower-bound for many point sets.

References

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

[2] John Iacono, *Key independent optimality*, Algorithmica, 42(1):3-10, 2005 *Key Independent Optimality*

[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.