Illinois Institute of Technology
Department of Computer Science

Lazy Weight-Balanced Search Trees

CS 535 Design and Analysis of Algorithms
Fall Semester, 2016

This is a solution to/elaboration of problem 17-3 on pages 473–474 of CLRS3.

Binary search trees are binary trees with the property that all descendants on the left of a node are smaller than the value in the node and all descendants on the right are greater than the value in the node. We want to keep the height of the binary search tree logarithmic in the number of items stored in the tree. There are many kinds of the balanced binary search trees, including red-black trees, height-balanced (AVL) trees, weight-balanced trees, 2-3 trees, and so on.

Here we do deletions and insertions in a binary search tree in a *lazy fashion*; that is, we rebalance rarely, but when we do, it is accomplished by a complete rebuilding of the tree (or subtree).

We assume that each node has fields for its *size* (the number of nodes in that subtree) and its *height* (depth of its deepest leaf). These fields can be eliminated without increasing the order of magnitude of the (amortized) costs of insertion and deletion—we explain how at the end of these notes. Note that the size and height functions are based on the number of nodes in the tree, *not* the number of items stored in the tree: we will do deletion in a lazy fashion by simply *marking* a node deleted but leaving it in the tree. We will not let more than half of the nodes be marked deleted, so if T is the tree with $\texttt{size}(T)$ nodes and contains $\texttt{num}(T)$ items (that is, unmarked nodes), we have

$$\texttt{size}(T) \geq \texttt{num}(T) \geq \texttt{size}(T)/2. \tag{1}$$

The degree of flexibility permitted in the tree is determined by a global parameter $\beta > 1$, the *balance factor*—the larger the value of β, the more elastic the shape of the tree. Each node can be marked or not, as indicated by a *mark bit*; the number of marked nodes for the entire tree, $m(T)$, is stored in another global parameter.

Balance property: Every node x in the tree must satisfy the height constraint

$$\texttt{height}(x) \leq \beta \lg \texttt{size}(x), \tag{2}$$

where $\texttt{height}(x)$ is the height the subtree rooted at node x and $\texttt{size}(x)$ is the number of nodes in the subtree rooted at x. This property guarantees $O(\log \texttt{num}(T))$ search times in a tree T. We need to examine the amortized cost of insertions and deletions.

Deletion of a node x is done lazily by marking the node x deleted, but leaving it in place—if a search "finds" a marked node, it reports the value is *not* in tree. When half the nodes in the tree are marked as having been deleted, the unmarked nodes of the tree are completely rebuilt into a perfectly balanced tree of height $\lceil \lg \texttt{num}(T) \rceil$ with no marked nodes. We explain below how to rebuild the tree in time $\Theta(\texttt{num}(T))$.

Insertion is done as usual: following the (unsuccessful) search path down the tree, the new node is inserted at the point where the search fails (that is, the null pointer is replaced by a pointer to the new node containing the inserted item). Assuming either that we have parent pointers or that the path we followed down the tree is stored in a stack as we descend, we then go backward up the path to from the newly inserted node toward the root of the tree to find the lowest node x (if any) such that

$$\texttt{height}(x) > \beta \lg \texttt{size}(x),$$

violating (2).

If such an x is found, the subtree rooted at x is completely rebuilt (as we said above, this takes time $\Theta(\texttt{num}(x))$; see below) into a perfectly balanced subtree of height $\lceil \lg \texttt{size}(x) - 1 \rceil$ (why -1?). Because this rebuilding reduces the height of that subtree by at least 1 (why?), all ancestor nodes satisfy the balance property (because they did before the insertion).

Claim: The amortized time for insertion/deletion in a lazy weight-balanced tree T is $O(\log \texttt{num}(T))$.

We prove this claim by the potential method. For each node x, define the *imbalance* at node x as

$$I(x) = \max\{0, |\texttt{size}(\texttt{left}(x)) - \texttt{size}(\texttt{right}(x))| - 1\},$$

and the potential function for the tree T as

$$\Phi(T) = \hat{c}m(T) + c \sum_{x \in T} I(x),$$

where the constants c and $\hat{c}$ will be specified later. This definition is similar to the definition of potential in problem 17-3 of CLRS3.

Amortized Cost of Deletion

Amortized cost is the sum of the actual cost and change in potential,

$$\Delta\Phi(T) = \Phi_{\text{after}}(T) - \Phi_{\text{before}}(T).$$

1. If no rebuilding is needed: The height constraint (2), together with (1) tells us that

   $$\text{Actual Cost} = O(\texttt{height}(T)) = O(\log \texttt{size}(T)) = O(\log \texttt{num}(T))$$

   for the search, and

   $$\Delta\Phi(T) = \hat{c} = O(1),$$

   since $m(T)$ increases by 1.

2. If the tree is rebuilt:

   $$\text{Actual Cost } = \Theta(\texttt{size}(T)), \tag{3}$$

   which is $\Theta(\texttt{num}(T))$ by (1), including the $O(\log \texttt{num}(T))$ search and the $\Theta(n)$ rebuilding—we explain below how to rebuild the tree in time $\Theta(\texttt{num}(T))$. Now,

   $$\Delta\Phi(T) \leq -\hat{c} \times \texttt{size}(T)/2 - I(T)$$

   because rebuilding occurs when half the nodes are marked, and after the rebuilding all nodes are unmarked and all nodes have imbalance 0. We choose $\hat{c}$ in the potential function so that $\hat{c}/2$ is larger than constant in $\Theta(\texttt{size}(T))$ in (3), the rebuilding process, giving an amortized time of $O(1)$ when the tree is rebuilt.

The amortized cost of deletion is thus $O(\log \texttt{num}(T))$.

Amortized Cost of Insertion

Again, the amortized cost is the sum of the actual cost and change in potential.

1. If no node becomes unbalanced, there is no rebuilding, so the height constraint (2) tells us, by (1), that

$$\text{Actual Cost } = O(\texttt{height}(T)) = O(\log \texttt{size}(T)) = O(\log \texttt{num}(T))$$

   for the search. The imbalance of each node along the path from the root to the site of the insertion can increase by at most one, so again the height constraint (2) tells us that

$$\Delta\Phi(T) \le \texttt{height}(T) \le \beta \lg \texttt{size}(T) = O(\log \texttt{num}(T)),$$

   giving an amortized cost of $O(\log \texttt{num}(T))$.

2. Suppose we have to rebuild at some node x because it became unbalanced—that is, because $\texttt{height}(x) > \beta \lg \texttt{size}(x)$ after insertion; after the rebuilding the imbalance $I(x)$ will be 0. The actual cost is $O(\log \texttt{size}(x))$ for the search portion and $O(\texttt{size}(X)) = O(\texttt{num}(x))$ for the rebuilding. To get the change in potential we need to calculate the imbalance $I(x)$ before the insertion. Assume, without loss of generality, that the insertion takes place in the left subtree of x; this means that before the insertion the left subtree of x was at least as tall as the right subtree,

$$\texttt{height}(\texttt{left}(x)) \ge \texttt{height}(\texttt{right}(x)),$$

   so that both before and after the insertion

$$\texttt{height}(x) = 1 + \texttt{height}(\texttt{left}(x)).$$

   But by the height constraint (2), because the tree was balanced below x before the insertion, it was also balanced at $\texttt{left}(x)$, so

$$\texttt{height}(\texttt{left}(x)) \le \beta \lg \texttt{size}(\texttt{left}(x)).$$

   After the insertion the tree is unbalanced at x, so now

$$\beta \lg \texttt{size}(x) \le \texttt{height}(x) = \texttt{height}(\texttt{left}(x)) + 1 \le \beta \lg \texttt{size}(\texttt{left}(x)) + 1.$$

   Because x is the lowest point of imbalance, both before and after the insertion we have

$$\texttt{height}(x) = \texttt{height}(\texttt{left}(x)) + 1 \le \beta \lg \texttt{size}(\texttt{left}(x)) + 1;$$

   Exponentiating this gives

$$2^{\beta \lg \texttt{size}(x)} \le 2^{\beta \lg \texttt{size}(\texttt{left}(x))+1},$$

   or

$$\texttt{size}(x)^{\beta} < 2\texttt{size}(\texttt{left}(x))^{\beta},$$

   so that after insertion

$$\texttt{size}(x) < 2^{1/\beta}\texttt{size}(\texttt{left}(x)). \tag{4}$$

   Hence before insertion (that is, using the old `size` values),

$$\texttt{size}(x) + 1 < 2^{1/\beta}(\texttt{size}(\texttt{left}(x)) + 1),$$

   or

$$\texttt{size}(x) < 2^{1/\beta}\texttt{size}(\texttt{left}(x)) + (2^{1/\beta} - 1).$$

Of course,

$$\texttt{size}(x) = \texttt{size}(\texttt{left}(x)) + \texttt{size}(\texttt{right}(x)) + 1,$$

before and after insertion, so before insertion

$$\begin{aligned}
\texttt{size}(\texttt{right}(x)) &= \texttt{size}(x) - \texttt{size}(\texttt{left}(x)) - 1 \\
&< 2^{1/\beta}\texttt{size}(\texttt{left}(x)) + (2^{1/\beta} - 1) - \texttt{size}(\texttt{left}(x)) - 1 \\
&< (2^{1/\beta} - 1)\texttt{size}(\texttt{left}(x)) + 2^{1/\beta} - 2 \\
&\leq \texttt{size}(\texttt{left}(x)),
\end{aligned}$$

because $\beta > 1$ means that $2^{1/\beta} - 1 < 1$ and $2^{1/\beta} - 2$ is negative. Hence before the insertion the definition of the imbalance $I(x)$ gives

$$\begin{aligned}
I(x) &\geq \texttt{size}(\texttt{left}(x)) - \texttt{size}(\texttt{right}(x)) - 1 \\
&> \frac{\texttt{size}(x)}{2^{1/\beta}} - \left[\left(1 - \frac{1}{2^{1/\beta}}\right)\texttt{size}(x) - 1\right] - 1 \\
&= \left[2^{1-1/\beta} - 1\right]\texttt{size}(x) \\
&= \Theta(\texttt{size}(x))
\end{aligned}$$

by (4) and because $\beta > 1$ implies $2^{1-1/\beta} > 1$.

We can now calculate the change in potential (and hence the amortized cost): After the insertion-with-rebuilding $I(x) = 0$, which reduces the potential by $\Theta(\texttt{size}(x))$; the rebuilding leaves all imbalances in the subtree rooted at x at 0. On the other hand, the rebuilding at x can leave the subtree rooted at x one level taller, increasing by 1 the imbalances of all ancestors of x; because the tree was balanced before the insertion, this is an increase in potential of at most $O(\texttt{height}(T)) = O(\log \texttt{size}(T))$. The rebuilding eliminates all marked nodes in the subtree rooted at x, a decrease in potential of at most $\texttt{size}(x)/2 = \Theta(\texttt{size}(x))$. The overall change in potential is therefore

$$\Delta\Phi(T) \leq c[-\Theta(\texttt{size}(x)) + \Theta(\log \texttt{size}(T))],$$

and the actual cost of the insertion-with-rebuilding is $\Theta(\log \texttt{size}(T))$ for the search/insertion portion and $\Theta(\texttt{size}(x))$ for the rebuilding portion. We can choose c in the potential function so that the pieces proportional to $\texttt{size}(x)$ cancel, leaving an amortized cost of

$$O(\log \texttt{size}(T)) = O(\log n).$$

Refinements

Rebuilding in linear time: Do an inorder traversal of the tree to produce an ordered array of the tree elements (CLRS3, page 288), after which a simple divide-and-conquer algorithm can build the perfectly balanced binary search tree (put the middle element at the root and construct the left and right subtrees recursively).

Eliminating the size and height fields: Change the rebalancing strategy so that rebalancing only occurs if the root of the tree becomes unbalanced—that is, if the insertion causes the height of the tree to violate (2). Rebalancing will still be done at the lowest unbalanced node, u along the path from the root to the newly inserted item x. If we keep the overall size and height of the tree as global parameters, and keep track of the length of the path as we descend in the tree during an insertion, we know at the point of insertion whether the tree violates the height constraint at the root, meaning there must be a rebuilding needed somewhere along the path. We now go upward along the path from x to the root, calculating the height of each subtree and its size as we go, using the formulas

$$\texttt{height}(\texttt{parent}(z)) = 1 + \max(\texttt{height}(z), \texttt{height}(\texttt{sibling}(z))),$$

and

$$\texttt{size}(\texttt{parent}(z)) = 1 + \texttt{size}(z) + \texttt{size}(\texttt{sibling}(z)),$$

checking that $\texttt{height}(\texttt{parent}(z)) \le \beta \lg \texttt{size}(\texttt{parent}(z))$. The process starts with $z = x$ and ends when we find an unbalanced node. The cost of doing this is $O(\texttt{size}(x))$, which is also the cost of the rebuilding of the subtree rooted at x.

References

[1] Galperin, Igal and Ronald L. Rivest, "Scapegoat trees," *Proc. Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 165–174, 1993.

[2] Andersson, Arne, "General balanced trees," *J. Algorithms* vol. 30 (1999), pp. 1–28.