18.415/6.854 Advanced Algorithms	October 29, 2001
Lecture 13	
Lecturer: Michel X. Goemans	*Scribe: Naveen Sunkavally*

1 Introduction

In the last lecture we introduced splay trees and defined the three different types of splay steps used to splay an element (i.e. bring it to the root). The three types (shown below as operating on node X) are:

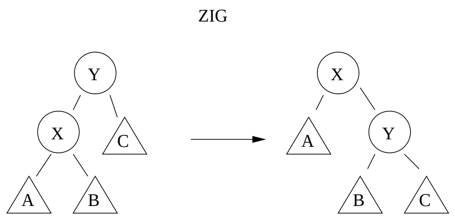

Figure 1: The parent of X is the root

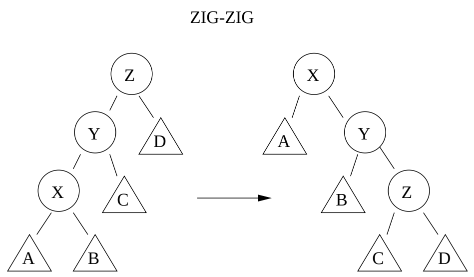

Figure 2: X and its parent are both left (or right) children

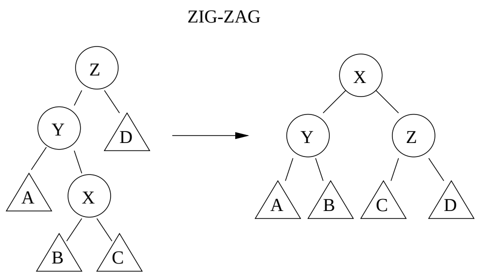

Figure 3: X is a right child and its parent is a left child (or vice versa)

2 Amortized Analysis of Splay Tree Operations

In this lecture we use amortized analysis to obtain bounds on the running times of splay operations. The analysis will show that the amortized cost of any splay tree operation (find, insert, delete, etc.) is of the order $O(log(n))$, where n is the number of nodes in the tree. Any single operation on a splay tree may take $O(n)$ time, but it also tends to make the tree more balanced, so that over time the average cost of any operation is $O(log(n))$.

The analysis uses the potential method (see CLR, chapter 18 for a description) to obtain the $O(log(n))$ bound. The first step is to define a weight, sum, and rank function at each node:

- Every node X has a weight $w(X) \geq 0$
- $s(X) = \sum_{Y \epsilon subtree(X)} w(Y)$
- $r(X) = log_2(s(X))$

We define the potential on the entire splay tree data structure at any moment in time i as the sum of all the ranks in the tree:

$$\phi(i) = \sum_{X \epsilon tree} r(X)$$

The potential is a measure of how balanced the tree is: trees with low potential for a given number of nodes are well balanced, while trees with high potential are poorly balanced. The amortized cost of a splay operation is then given by the actual cost of the operation plus the change in potential of the tree. Operations whose actual costs are high should come with the benefit of lowering the potential of the tree so that the amortized cost stays reasonable.

2.1 Amortized Cost of a Splay Step

The following lemma gives the amortized cost of a single splay step:

Lemma 1 *Let $r(X)$ be the rank of X before a splay step, and let $r'(X)$ be the rank of X after a splay step. Then the amortized cost of the splay step shown in figure 1 (ZIG) is $\leq 3(r'(X) - r(X)) + 1$. The amortized cost of the splay steps shown in figure 2 and 3 (ZIG-ZIG and ZIG-ZAG) is $\leq 3(r'(X) - r(X))$.*

Proof of Lemma 1: Consider each of the cases separately:

In **case 1** (ZIG), we have

$$\begin{aligned} \text{Amortized cost} &= \text{Actual cost } + \phi(i+1) - \phi(i) & (1) \\ &= 1 + (r'(X) + r'(Y) - r(X) - r(Y)) & (2) \end{aligned}$$

The actual cost of the splay step in this case is 1 because we only perform one rotation to bring X to the root. Because $r'(X) = r(Y)$, and because $r'(Y) \leq r'(X)$, we get:

$$\begin{aligned} \text{Amortized cost} &\leq 1 + r'(X) - r(X) & (3) \\ &\leq 1 + 3(r'(X) - r(X)) & (4) \end{aligned}$$

In **case 2** (ZIG-ZIG), the actual cost is 2 because we are performing a double rotation. We can take advantage of the fact that $r'(X) = r(Z)$, $r(Y) \geq r(X)$, and $r'(Y) \leq r'(X)$ to arrive at:

$$
\begin{aligned}
\text{Amortized cost} &= \text{Actual cost } + \phi(i+1) - \phi(i) && (5)\\
&= 2 + (r'(X) + r'(Y) + r'(Z) - r(X) - r(Y) - r(Z)) && (6)\\
&\leq 2 + r'(X) + r'(Z) - r(X) - r(X) && (7)
\end{aligned}
$$

To simplify further, we take advantage of the fact that the *log* function is concave, so that for any two points a and b, a and $b > 0$, it must be the case that $\frac{log_2(a)+log_2(b)}{2} \leq log_2(\frac{a+b}{2})$. In figure 2, notice that $s(X) + s'(Z) \leq s'(X)$, or equivalently $\frac{s(X)+s'(Z)}{2} \leq \frac{s'(X)}{2}$. By the concavity of the log function, $\frac{r(X)+r'(Z)}{2} \leq log_2(\frac{s(X)+s'(Z)}{2})$, which by transitivity implies that $\frac{r(X)+r'(Z)}{2} \leq log_2(\frac{s'(X)}{2}) = r'(X) - 1$, and $r'(Z) \leq 2r'(X) - 2 - r(X)$. Thus,

$$
\begin{aligned}
\text{Amortized cost} &\leq 2 + r'(X) + (2r'(X) - 2 - r(X)) - r(X) - r(X) && (8)\\
&\leq 3(r'(X) - r(X)) && (9)
\end{aligned}
$$

In **case 3** (ZIG-ZAG), the actual cost of the double rotation is 2, and we again use the fact that $r'(X) = r(Z)$ and $r(Y) \geq r(X)$ to state:

$$
\begin{aligned}
\text{Amortized cost} &= \text{Actual cost } + \phi(i+1) - \phi(i) && (10)\\
&= 2 + (r'(X) + r'(Y) + r'(Z) - r(X) - r(Y) - r(Z)) && (11)\\
&\leq 2 + r'(Y) + r'(Z) - r(X) - r(X) && (12)
\end{aligned}
$$

From figure 3, it is evident that $s'(Y)+s'(Z) \leq s'(X)$, and using the concavity of the *log* function, we get that $r'(Y) + r'(Z) \leq 2(r'(X) - 1)$. Thus,

$$
\begin{aligned}
\text{Amortized cost} &\leq 2 + (2r'(X) - 2) - r(X) - r(X) && (13)\\
&\leq 3(r'(X) - r(X)) && (14)
\end{aligned}
$$

□

2.2 Amortized Analysis of a Splay Operation

The following two corollaries follow from the above analysis of the amortized running time of a single splay step.

Corollary 2 *Let V be the vertex of the splay tree before a splay operation is carried out. Then the amortized cost of splaying a node X is $O(log(n))$.*

Proof of Corollary 2: The amortized cost of splaying at X equals the sum of the amortized costs of all the single splay steps. This sum telescopes to yield that the amortized cost of splaying X is $\leq 3(r(V) - r(X)) + 1$. The extra constant term 1 accounts for the possibility of a case 1 rotation.

Note that this analysis is independent of the chosen weights in the tree. Suppose we set all weights to be 1. Then the maximum possible difference between $r(V)$ and $r(X)$ equals the *log* of number of nodes in the tree. So the amortized cost of splaying X is $O(log(n))$. □

Corollary 3 *The actual cost of a sequence of m splayings is $O((m+n)log(n))$.*

Proof of Corollary 3: The actual cost equals the amortized cost plus the gain from the credit invariant. Or equivalently, if i corresponds to the time before the m splayings, and $i+m$ corresponds to the time after the m splayings, we get:

$$\text{Actual cost of } m \text{ splays} \;=\; \text{Amortized cost of } m \text{ splays } + \phi(i) - \phi(i+m)$$

From corollary 2, the amortized cost of m splays is $O(mlog(n))$. The maximum change in potential, again assuming all weights are 1, is $O(nlog(n))$, so the actual cost is $O((m+n)log(n))$. $\square$

We have shown that the amortized cost of splaying is $O(log(n))$, but we have yet to show that the cost of operations such as find or insert are also $O(log(n))$. Most operations, e.g. find, which do not modify the tree are clearly within a constant factor of splaying operation, because we can argue that the cost, say, of finding an element is roughly twice the cost of splaying that element. Thus most operations have an amortized cost of $O(log(n))$. This analysis however does not necessarily follow for operations such as insert, which modify the tree.

It can be shown that the insert operation also has an amortized running time of $O(log(n))$ by considering how much the potential of the tree increases right after a new node is added (prior to the subsequent splay that brings it to the root). If the potential increases by a factor of $O(log(n))$, then the total amortized cost of the insert operation is also $O(log(n))$. This is indeed the case, as can be shown by the following analysis:

Consider an insert operation on the node X. After X has been inserted (and right before the splay), all nodes Y_j on the path from the root to X witness an increase in their $s(Y_j)$ by 1, or equivalently the rank for each Y_j increases by $log_2(\frac{s(Y_j)+1}{s(Y_j)})$. Let Y_1 correspond to the parent of X, Y_2 correspond to the grandparent of X, and so on until we get to Y_k, which corresponds to the root node. The increase in the total rank is given by:

$$\begin{aligned}
\text{Increase in } \phi(i) &= log_2(\frac{s(Y_1)+1}{s(Y_1)}) + log_2(\frac{s(Y_2)+1}{s(Y_2)}) + log_2(\frac{s(Y_3)+1}{s(Y_4)})...log_2(\frac{s(Y_k)+1}{s(Y_k)}) && (15)\\
&= log_2(\frac{s(Y_1)+1}{s(Y_1)}\frac{s(Y_2)+1}{s(Y_2)}\frac{s(Y_3)+1}{s(Y_3)}...\frac{s(Y_k)+1}{s(Y_k)}) && (16)
\end{aligned}$$

Next we make use of the fact that $s(Y_i)+1 \leq s(Y_{i+1})$ to telescope the product contained in the logarithm:

$$\begin{aligned}
\text{Increase in } \phi(i) &= log_2(\frac{s(Y_1)+1}{s(Y_1)}\frac{s(Y_2)+1}{s(Y_2)}\frac{s(Y_3)+1}{s(Y_3)}...\frac{s(Y_k)+1}{s(Y_k)}) && (17)\\
&\leq log_2(\frac{s(Y_k)+1}{s(Y_1)}) \leq log_2(n) && (18)
\end{aligned}$$

The increase in potential is $\leq log_2(n)$, so the amortized cost of the insert operation is indeed $O(log(n))$.

Other operations which modify the tree, such as delete, can also be shown to run in $O(log(n))$ time. In general, the performance of a splay tree is in fact so good that, for a given set of nodes, it can be shown that a splay tree performs to within a constant of the best static binary tree. This is called the *static optimality theorem*, which is stated below:

Theorem 4 *Suppose there are n objects that are to be repeatedly accessed. Each object is to be accessed at least once, and a total of m accesses are to be performed. The total running time of the m access operations using a splay tree is within a constant factor of the total running time taken using the best static binary search tree.*

The *dynamic binary search tree conjecture* goes further than the static optimality theorem.

Conjecture: *Suppose there are n objects that are to be repeatedly accessed. A total of m accesses are to be performed. The total running time of the m access operations using a splay tree is $O(n +$ the running time taken using the best dynamic binary search tree), i.e. the best binary search tree structure that allows rotations.*

3 Dynamic Trees

The dynamic trees data structure manages *a collection of disjoint (not necessarily binary) rooted trees.* A cost is associated with each node in each tree; this cost represents the cost on the edge to the node from the node's parent, unless the node is the root, in which case the cost is set to ∞ by default.

The dynamic trees data structure is useful for implementing admissible cycle cancellation, as will be shown in a later lecture. The data structure supports the following operations:

- $make-tree(V)$: Create a tree with a single node whose root is V. The cost of V is set to ∞.
- $find-root(V)$: Find and return the root of the tree containing the node V.
- $find-cost(V)$: Return the cost of the node V.
- $find-min(V)$: Return W, the node on the path from $find-root(V)$ to V with the minimum cost.
- $add-cost(x, V)$: Add x to the cost of all nodes on the path from $find-root(V)$ to V.
- $cut(V)$: Split the tree containing V into two trees by removing the edge between V and its parent. V becomes the root of a new tree, and its cost is set to ∞.
- $link(V, W, x)$: W is assumed to be the root of a tree which does not contain V. Merge the trees containing W and V by adding an edge between W and V. Set the cost of W equal to x.

Using splay trees, we shall see that the above operations can be made to run in $O(log(n))$ amortized time. The following theorem establishes this fact.

Theorem 5 *A sequence of m dynamic tree operations takes $O((m + n)log(n))$ time.*

Proof of Theorem 5: The first step of this proof is to decompose each tree into a set of node-disjoint paths. These node-disjoint paths are then viewed as splay trees, and the node on any node-disjoint path furthest away from the root of the original tree corresponds to the root of the splay tree.

This theorem will be proved in the next lecture.

□