21120491 Advanced Data Structure and Algorithm Analysis **Spring-Summer 2022**

Lecture 2: Splay Tree

Lecturer: Deshi Ye *Scribe: D. Ye*

2.1 Overview

Recall that in the AVL tree, we need to update the height or the balance factor in the structure, and always keep the balance of height between a node's left and right child. During the Rebalancing procedure, we need to calculate or update the height information along the path from the inserted or deleted node to the root. This updating consumes computation resources and takes time.

Can we define another balanced search tree without maintaining the height?

The splay tree was invented by Daniel Sleator and Robert Tarjan [1] in 1985. Splay tree is In 2000, Danny Sleator and Robert Tarjan won the ACM Kanellakis Theory and Practice Award for their papers on splay trees and amortized analysis.

Splay trees are used in Windows NT (in the virtual memory, networking, and file system code), the gcc compiler and GNU C++ library, the sed string editor, Fore Systems network routers, the most popular implementation of Unix malloc, Linux loadable kernel modules, and in much other software.

In the above applications, one common property is the locality that the same element will be accessed multiple times in the near future. Thus, it would be a benefit to put the data most recently access near the root.

In this section, we aim to design a well-balanced search tree for multiple operations.

Our target is to show that any M consecutive splay operations starting from an empty tree take at most $O(Mlogn)$ time, where n is the maximum number of nodes during these operations. In detail, we are going to show the amortized cost of such a splay operation in n-node tree is $O(\log n)$.

2.2 Splay tree

A splay tree is a "self-adjust" binary search tree. To access a node x in a splay tree, we use the standard binary search tree algorithm to find x, and then move the node x by splay operations to the root of the tree without breaking the binary search tree invariant.

The first idea is rotation, i.e., moving every accessed node to the root kept by the rotation operation. However, this idea does not work, such as access nodes from 1 to n sequentially in the splay tree that forms a chain, in which node 1 is the leave and n is the root, and a node i is a left child of node $i+1$. One can easily check that we need $O(n^2)$ time for these total n accesses.

Sleator and Tarjan [1] provide splay operations as below. The general idea of a splay operation is to move the node being splayed upward in the tree, and they also shorten the path to any nodes along the path to

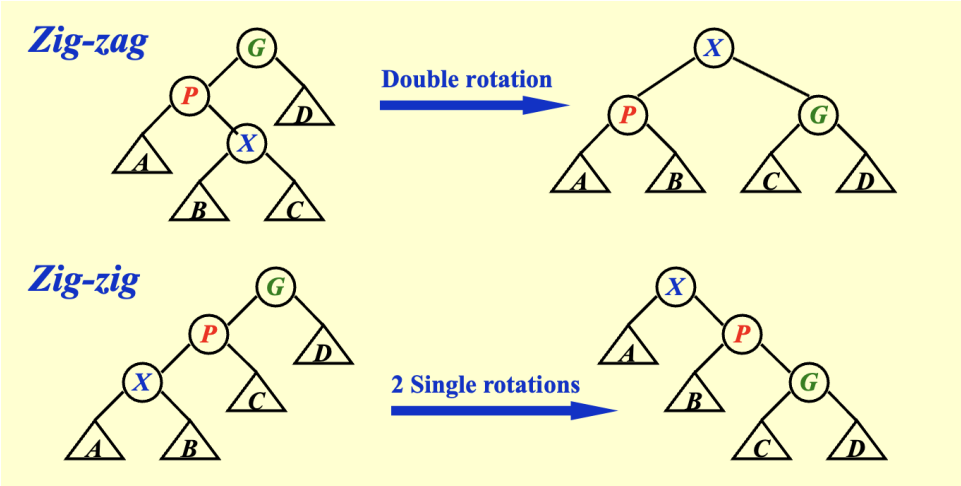

Figure 2.1: Zig-zag and Zig-zig

the splayed node.

For any non-root node X, denote its parent by P and grandparent by G.

Case 1: If P is the root, then make a rotation such that X is the root. (A single rotation is sufficient in this case). Otherwise, we kept doing one of the following two cases.

Case 2: Zig-zag, it is the case that Parent P and grandparent G are in different directions, i.e., the splayed node X is the left child of a right child or vice-versa. Perform a double rotation on the subtree root at G, we get a new subtree root at X, see for an illustration in Fig. 2.1.

Case 3: Zig-zig, it is the case that Parent P and grandparent G are in the same direction, i.e., the splayed node X is the left child of a left child or vice-versa (the right child of a right child ("zag-zag")). Perform two single rotations (the first rotation is to make the node G is a child of P, and the second rotation will guarantee that P is a child of X), see for an illustration in Fig. 2.1.

In other words, we define the $splay(X)$ operation as below.

Splaying a node X is equal to applying the zig-zig, zig-zag, and/or zig operation on the node X until node X becomes the root node of the binary (search) tree.

2.3 Amortized cost for Splay tree

A Splay tree would produce a balanced tree for repeated access operations. To get the strict proof of balancing, we adopt the concept of amortized analysis.

Amortized analysis is a worst-case bound on the average time per operation. We know that the splay operation that moving the target node X to root might require $O(n)$ in the worst case. But it costs 1 if we access it next time immediately. To get the worst-case analysis, the total cost of M operations is $O(Mn)$. But this analysis is too pessimistic. During these M consecutive operations, some operation costs large while some operations might cost tiny. The amortized analysis is to give the upper bound on these M consecutive operations. Of course $O(Mn)$ is one such an upper bound, but can we find a smaller upper bound?

In the following, we show that the total cost of M consecutive operations start from an empty is upper bounded by $O(M \log n)$, which implies that the amortized cost of a splay operation is $O(\log n)$.

We will use potential function Φ to analysis the amortized cost. Let $S(i)$ be the number of descendants of node i (the node i is included). Let $R(i)$ denote the rank of the node i, which is $R(i) = \log S(i)$. The potential of the splay tree $\Phi(T)$ is defined as the sum of the ranks of all the nodes in the tree T.

$$\Phi(T) = \sum_{i \in T} \log S(i) = \sum_{i \in T} R(i)$$

Lemma 2.1 *The amortized cost of a zig-zig or zig-zag operation on node X is at most of $3(R_2(X) - R_1(X))$, and the amortized cost of a zig operation on node X is at most of $3(R_2(X) - R_1(X)) + 1$, where $R_2(X)$ and $R_1(X)$ denote the rank of X before and after the splay step.*

Proof: We do the analysis case by case in detail. The cost of one single rotation is defined to be 1. Again, for any non-root node X, denote its parent by P and grandparent by G.

Let c and $\hat{c}$ be the actual cost and the amortized cost of the operation (zig-zig, zig-zag, or zig), respectively. Denote T_1 to be the tree before this splay operation and T_2 to be the tree after a splay operation.

Case 1: zig-zig. This operation actually costs 2, because 2 single rotation is required,i.e., $c = 2$. The amortized cost $\hat{c}$ is given as below. We see that only the rank of nodes X, P, and G changes by the splaying operation, while the rank of the other nodes remains the same.

$$\begin{aligned}
\hat{c} &= c + \Phi(T_2) - \Phi(T_1) \\
&= 2 + R_2(X) + R_2(P) + R_2(G) - (R_1(X) + R_1(P) + R_1(G)) \\
&\leq 2 + R_2(X) + R_2(G) - (R_1(X) + R_1(P)) \\
&\leq 2 + R_2(X) + R_2(G) - 2R_1(X)
\end{aligned}$$

The inequalities hold because of $R_2(P) \leq R_1(G)$ and $R_1(X) \leq R_1(P)$.

To show $2 + R_2(X) + R_2(G) - 2R_1(X) \leq 3(R_2(X) - R_1(X))$, we only need to show $R_1(X) + R_2(G) - 2R_2(X) \leq -2$. See an illustration in Fig. 2.1, let us abuse the notations A, B, C, D to denote also the size of those subtrees, respectively. Then $R_1(X) = \log(1 + A + B)$, and $R_2(G) = \log(1 + C + D)$, and $R_2(X) = \log(3 + A + B + C + D)$. By Lemma 2.2, one can check that the inequality $R_1(X) + R_2(G) - 2R_2(X) \leq -2$ satisfied.

In sum, $\hat{c} \leq 3(R_2(X) - R_1(X))$ for zig-zig case.

Case 2: zig-zag. This operation actually costs 2, because a double rotation is required,i.e., $c = 2$. The amortized cost $\hat{c}$ is given as below. We see that only the rank of nodes X, P, and G changes by the splaying operation, while the rank of the other nodes remains the same.

$$\begin{aligned}
\hat{c} &= c + \Phi(T_2) - \Phi(T_1) \\
&= 2 + R_2(X) + R_2(P) + R_2(G) - (R_1(X) + R_1(P) + R_1(G)) \\
&= 2 + R_2(P) + R_2(G) - R_1(X) - R_1(P) \\
&\leq 2 + R_2(P) + R_2(G) - 2R_1(X)
\end{aligned}$$

The above equation or inequality holds since $R_1(G) = R_2(X)$ and $R_1(X) \leq R_1(P)$. Similar as zig-zig case, we know that $R_2(P) = \log(1 + A + B)$, $R_2(G) = \log(1 + C + D)$, while $R_2(X) = \log(3 + A + B + C + D)$. Hence, we have $\hat{c} \leq 2 + R_2(P) + R_2(G) - 2R_1(X) \leq 2(R_2(X) - R_1(X))$ by Lemma 2.2.

Since $R_2(X) > R_1(X)$, we get that $\hat{c} \leq 3(R_2(X) - R_1(X))$.

Case 3: Zig. This operation actually costs 1, since one single rotation will move X to the root, i.e. $c = 1$. In this case, only the node X and P change their ranks by splaying.

$$\begin{aligned} \hat{c} &= c + \Phi(T_2) - \Phi(T_1) \\ &= 1 + R_2(X) + R_2(P) - (R_1(X) + R_1(P)) \\ &\leq 1 + R_2(X) - R_1(X) \end{aligned}$$

The last inequality holds since $R_2(P) < R_1(P)$. Moreover, $R_2(X) \geq R_1(X)$, we obtain that $\hat{c} \leq 1 + 3(R_2(X) - R_1(X))$. ■

Lemma 2.2 *For any positive integers a, b, c, we have $\log a + \log b - 2\log c \leq -2$ if $a + b \leq c$.*

Proof: Note that $\sqrt{ab} \leq \frac{a+b}{2} \leq \frac{c}{2}$, then $ab \leq \frac{c^2}{4}$, and the lemma holds by taking logarithm on both sides. ■

Theorem 2.3 *The amortized time to splay a tree with root T at node X is at most $3(R(T) - R(X)) + 1 = O(\log N)$, where N is the number of nodes in the tree.*

Proof: Recall that splaying a node X is is equivalent to applying the zig-zig, zig-zag, and/or zig operation on the node X until node X becomes the root node of the binary search tree. W.L.O.G. suppose that the operation $Splay(X)$ takes k steps, the final step is the zig operation while previous operations are zig-zig or zig-zag. Now, assume that we splay a node X and let $R_i(X)$ be the rank of X after the i-th splay step for $0 \leq i \leq k$.

By Lemma 2.1, and in each splay operation there can be at most one zig-step, the total amortized cost of the splay operation is at most

$$1 + \sum_{i=1}^{k} 3(R_i(X) - R_{i-1}(X)) = 1 + 3R_k(X) - 3R_0(X)$$

Given the definition of rank, $R_k(X) = R(T)$ because X is the root at the end of splaying operation, and $R_0(X)$ is the initial rank of X. Since $R(T) = \log N$, we have the amortized cost of a splay operation is $O(\log N)$. ■

When we access a node X, we need to find such node X and then splay X to the root. We have already shown that the cost of splay operation is $O(\log N)$.

Question: What is the amortized cost of finding the node X?

Answer: the cost of finding the node X is proportional to the cost of splaying X to the root. Because the zig-zig or zig-zag operation will make the node X moves 2 levels up the binary search tree, and the zig operation will move 1 level up the binary search tree. Hence, the cost of finding X is equivalent to the depth of node X, which is proportional to the number of splay steps in the splaying operation. Hence, the amortized cost of finding node X is also $O(\log N)$.

Remark: Any M consecutive tree operations starting from an empty tree take at most $O(MlogN)$ time. We would like to do M consecutive splay operations. Let T_0 be the initial tree, and T_i be the tree after ith splay operation, where $1 \leq i \leq M$.

Let c_i be the actual cost of ith splay operation, and $\hat{c}_i$ be the amortized cost of the ith splay operation. Then $\hat{c}_i = c_i + \Phi(T_i) - \Phi(T_{i-1})$. The amortized cost of M consecutive tree operations is

$$\sum_{i=1}^{M} \hat{c}_i = \sum_{i=1}^{M} c_i + \Phi(T_M) - \Phi(T_0).$$

The total cost of M consecutive tree operations is $\sum_{i=1}^{M} c_i = \sum_{i=1}^{M} \hat{c}_i + \Phi(T_0) - \Phi(T_M) \leq M \log N + N \log N = O(M + N) \log N$. Because the initial potential $\Phi(T_0) \leq N \log N$. If the initial tree T_0 is empty, then $\Phi(T_0) = 0$, we have $\sum_{i=1}^{M} c_i \leq \sum_{i=1}^{M} \hat{c}_i = O(M \log N)$.

2.3.1 Insert in Splay tree

In order to insert a node into a splay tree, we perform standard binary search tree insertion and then splay on the inserted node.

We have shown that the amortized cost of splaying operation is $O(\log N)$. After insertion of a new node, we must also account for the increase in potential when we insert the node.

Assume the node X is inserted at depth of k. Let Y_1 be the parent of X, and Y_2 be the parent of Y_1, and so on the root is Y_k. The potential changes only along the path from the node X to the root. Let us denote by $\Delta(\Phi)$ the increase of potential. Let $R(X)$ and $R'(X)$ be the rank of node X before insertion and after the insertion, respectively. The increase of potential is given as below.

$$\begin{aligned}
\Delta(\Phi) &= \sum_{i=1}^{k} (R'(Y_i) - R(Y_i)) \\
&= \sum_{i=1}^{k} (\log(s(Y_i) + 1) - \log(s(Y_i))) \\
&= \sum_{i=1}^{k} \log \frac{s(Y_i) + 1}{s(Y_i)} \\
&= \log(\prod_{i=1}^{k} \frac{s(Y_i) + 1}{s(Y_i)}) \\
&\leq \log(\frac{s(Y_2)}{s(Y_1)} \frac{s(Y_3)}{s(Y_2)} \cdots \frac{s(Y_k) + 1}{s(Y_k)}) \\
&= \log \frac{s(Y_k) + 1}{s(Y_1)} \\
&\leq \log N
\end{aligned}$$

In above formulations, we have $s(Y_i) + 1 \leq s(Y_{i+1})$.

By counting the increase of potential, the amortized cost of the insertion is $O(\log N)$.

References

[1] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, jul 1985.