

Lecture 5: Splay Trees

Lecturer: Gary Miller

Scribe: Jiayi Li, Tianyi Yang

1 Introduction

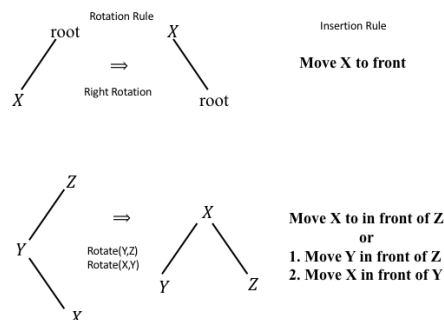
Recall from previous lecture that balanced binary search trees (BST) are those whose max depths are $O(\log n)$. While this property guarantees balanced BST to have $O(\log n)$ worst-case performance, sometimes the worst-case performance may not be of the most interest. Indeed, suppose a worst-case query is searched multiple times, ideally we would want the data structure to adjust itself to accommodate the observed empirical frequency so that when the same query is searched again, a faster time can be achieved. The splay tree is a variant of BST that does exactly that. Every time $\text{SEARCH}(x)$ is called on a splay tree, (assume x is indeed in the tree), in addition to the searching, the splay tree also rotates x to the root using a specialized function $\text{SPLAY}(x)$.

Although splay trees are not balanced according to the worst-case standard, they are “self-balancing” thanks to $\text{SPLAY}(x)$ in that no information is stored to maintain the balance. In the long run, the amortized complexity works out to $O(\log n)$ per operation.

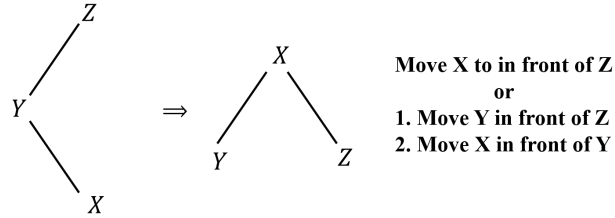
2 Splay(x)

As mentioned above, the distinguished feature of splay trees is the function $\text{SPLAY}(x)$. $\text{SPLAY}(x)$ rotates a node x to the root using a series of three rules, which are described below. We give to definitions of these rules. The first is an imperative description using rotations, while the second describes the new BST as a change to an insertion order or priority order that determines the original tree. Note that an element can be moved up in a BST by moving it earlier in the insertion order.

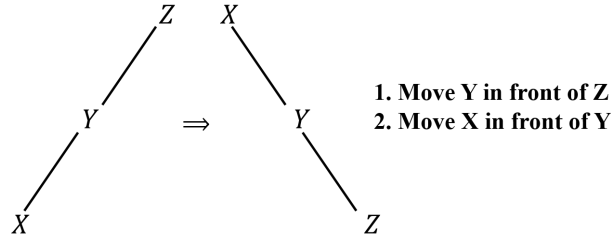
1. Zig



2. Zig-Zag



3. **Zig-Zig** (the interesting case)



Depending on the relative positions of x and its $parent(x)$ and $grandparent(x)$, one of the above rules will dictate how the rotation takes place. x will keep rotating up until it becomes the root.

In addition to a set of rotation rules, $SPLAY(x)$ can also be described by rules of changing priorities for a treap.

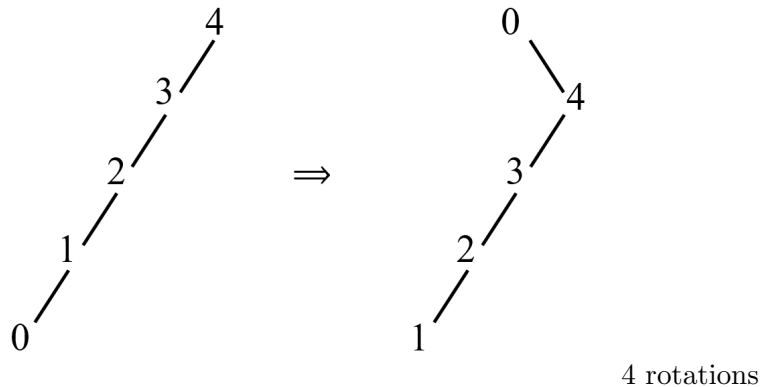
Priority version of Splay: Given x, y, z such that $parent(x) = y$ and $parent(y) = z$.

- move y in front of z
- move x in front of y

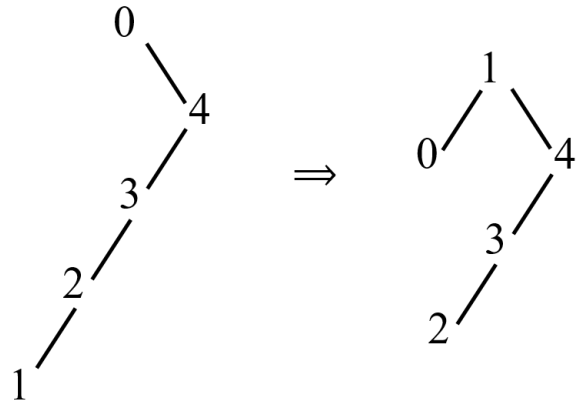
One might ask why it is not sufficient to rotate x with $parent(x)$. In other words, why include the Zig-Zig rule involving $grandparent(x)$? This can be illustrated through an example.

Example 2.1. Let the keys be 0, 1, 2, 3, 4 and the priorities be (4, 3, 2, 1, 0). Suppose we do not include the Zig-Zig rule and simply rotate x with $parent(x)$ each time. Consider the following sequence of operations.

- **Move 0 to the front (MTF 0)** Priorities: (4, 3, 2, 1, 0) \Rightarrow (0, 4, 3, 2, 1)

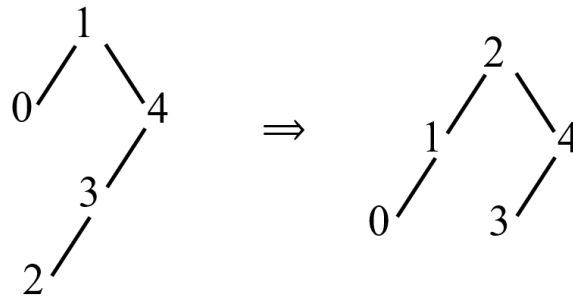


- **MTF 1** Priorities: (0, 4, 3, 2, 1) \Rightarrow (1, 0, 4, 3, 2)



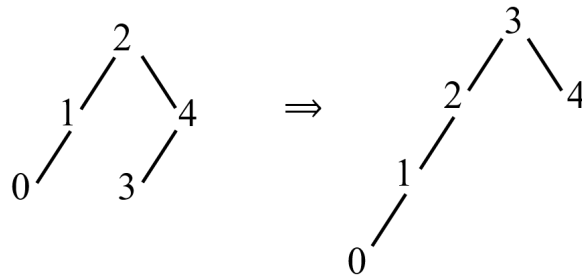
4 rotations

- **MTF 2** Priorities: $(1, 0, 4, 3, 2) \Rightarrow (2, 1, 0, 4, 3)$



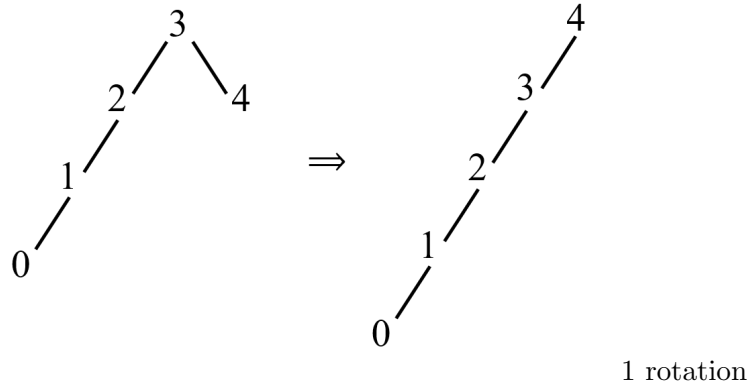
3 rotations

- **MTF 3** Priorities: $(2, 1, 0, 4, 3) \Rightarrow (3, 2, 1, 0, 4)$



2 rotations

- **MTF 4** Priorities: $(3, 2, 1, 0, 4) \Rightarrow (4, 3, 2, 1, 0)$

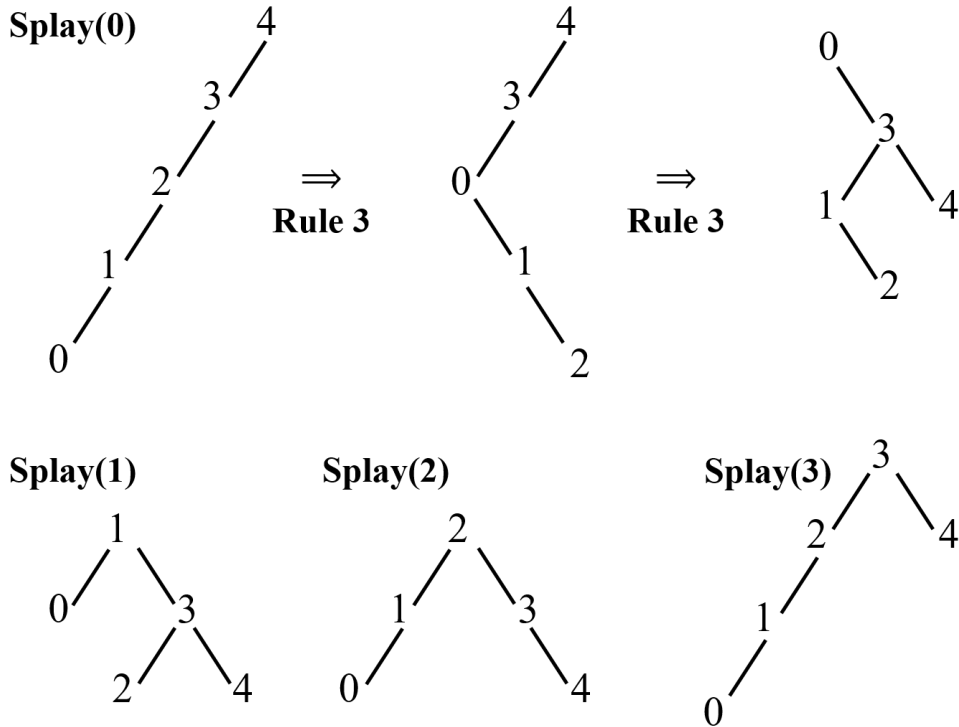


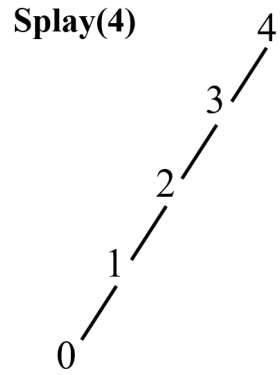
The total number of rotations is $4 + 4 + 3 + 2 + 1 = 14$.

In general, given key $(n, n - 1, \dots, 0)$ and insertion order $(n, n - 1, \dots, 0)$ if we leave out the Zig-Zig rule and splay $0, \dots, n$ in this order we get the following number of rotations performed. Thus the average number of rotation: $\theta(n)$.

Operation	MTF(0)	MTF(1)	MTF(2)	...	MTF(n)	$n + 1$ operations
rotations	n	n	$n - 1$...	1	$\Omega(n^2)$

On the other hand, if we include the Zig-zig rule, then the resulting trees would look as follows:





The total number of rotations is $6 + 2 + 2 + 1 + 1 = 12 < 14$.

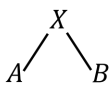
3 Splay Dictionary Operations

We next show how one can use Splay to implement an efficient dictionary. The basic operations of splay trees all employ $\text{SPLAY}(x, T)$.

Algorithm 1 Splay Dictionary Operations


```
function INSERT( $x, T$ )  
    VANILLAININSERT( $x, T$ )  
    SPLAY( $x, T$ )  
end function
```

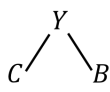
```
function DELETE( $x, T$ )
```

```
    SPLAY( $x, T$ ), giving 
```

```
    if  $B$  is empty then  
        return  $A$ 
```

```
    else
```

```
        SPLAY( $largest(A), A$ ), giving 
```

```
        return 
```

```
    end if
```

```
end function
```

```
function LOOKUP( $x, T$ )
```

```
    SEARCH( $x, T$ )
```

```
    if  $x$  is found then
```

```
        return SPLAY( $x, T$ )
```

```
    else
```

```
        return SPLAY( $parent(x), T$ )
```

```
    end if
```

```
end function
```

Note that $SPLAY(x, T)$ is called even if $LOOKUP(x)$ fails to find the query. This is because splay trees are designed to change all the time precisely to address the problem discussed in the introduction.

4 Amortized Analysis of Splay

4.1 Preliminaries

We use potential method to study the amortized cost of $SPLAY$. In order to do so, we first need to define a potential function $\Phi(T)$. Conceptually, $\Phi(T)$ should be large for unbalanced trees.

Definition 4.1. Let T be a binary search tree, $S(x)$ is the number of nodes in subtree rooted at x .

Definition 4.2. Let T be a binary search tree, the rank of node x is

$$r(x) = \lfloor \log_2(S(x)) \rfloor$$

Definition 4.3. The potential of a binary search tree T is given by

$$\Phi(T) = \sum_{x \in T} r(x)$$

Example 4.4. If T is a line, then

$$\Phi(T) = \sum_{i=1}^n \lceil \log_2(i) \rceil \approx \sum_{i=1}^n \log_2(i) = \log(n!) = \Theta(n \log(n))$$

To see the last equality, first note that

$$\log(n!) < \log(n^n) = n \log(n) \Rightarrow \log(n!) \in O(n \log(n))$$

In addition,

$$\begin{aligned} \log(n!) &= \sum_{i=1}^n \log(i) > \sum_{i=\frac{n}{2}}^n \log(i) > \sum_{i=\frac{n}{2}}^n \log\left(\frac{n}{2}\right) = \left(\frac{n}{2}\right) \log\left(\frac{n}{2}\right) \\ &\Rightarrow \log(n!) \in \Omega(n \log(n)) \end{aligned}$$

Hence $\log(n!) = \Theta(n \log(n))$.

Example 4.5. If T is balanced, then $\Phi(T) = \Theta(n)$

Claim 4.6. For all T , $\Phi(T) = \Omega(n)$ and $\Phi(T) = O(n \log n)$. That is, lines and balanced trees provide upper and lower bounds of the complexity of the trees.

Given the potential function, we are ready to defined the amortized cost (AC):

$$\text{AC} = \text{Unit-Cost} + \Delta\Phi$$

where

$$\text{Unit-Cost} \equiv \text{Number of rule applications} = \lceil \frac{\text{depth}}{2} \rceil = \frac{\text{Number of rotations}}{2}$$

4.2 Access Lemma

In order to study the amortized cost of searching in a splay tree, we first need to understand the run time of SPLAY. The Access Lemma is the main result in this regard. We first state the Access Lemma.

Lemma 4.7 (Access Lemma). $AC(\text{SPLAY}(x)) \leq 3(r(\text{root}) - r(x)) + 1$

Corollary 4.8. $AC(\text{SPLAY}(x)) \leq 3 \log n + 1$

The proof of Access Lemma requires some preliminaries results regarding the rank, which we will show first.

Lemma 4.9 (Simple Fact 1). If node x has two children a, b such that $r(a) = r(b) = r$, then $r(x) = r' > r$.

Proof.

$$\begin{aligned} S(x) &= S(a) + S(b) + 1 > S(a) + S(b) \geq 2^{r(a)} + 2^{r(b)} = 2^r + 2^r = 2^{r+1} \\ &\Rightarrow r(x) \geq r + 1 \end{aligned}$$

□

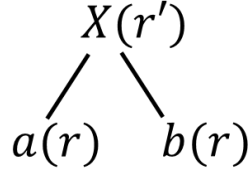


Figure 1: Simple Fact 1. $r' > r$

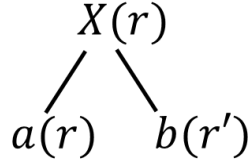


Figure 2: Simple Fact 2. $r > r'$

Corollary 4.10 (Simple Fact 2). *If node x has two children a, b such that $r(x) = r(a)$, then $r(x) > r(b)$*

Corollary 4.11 (Simple Fact 3). *If node x has two children a, b such that $r(x) = r(a)$, then $2r(x) > r(a) + r(b)$*

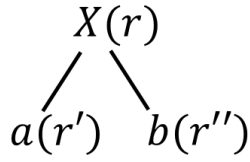


Figure 3: Simple Fact 2. $2r > r' + r''$

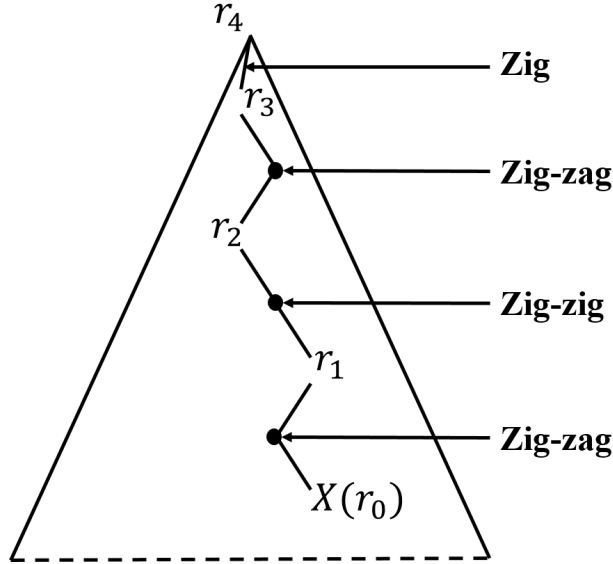
Proof of Access Lemma

Proof. Throughout the proof, we will let C, P, GP denote *child, parent, grandparent*, respectively. We claim that it is sufficient to show that

$$AC(\text{Zig}) \leq 3(r(\text{root}) - r(C)) + 1 \tag{1}$$

$$AC(\text{Zig-zag}), AC(\text{Zig-zag}) \leq 3(r(GP) - r(C)) \tag{2}$$

This can be illustrated through an example.

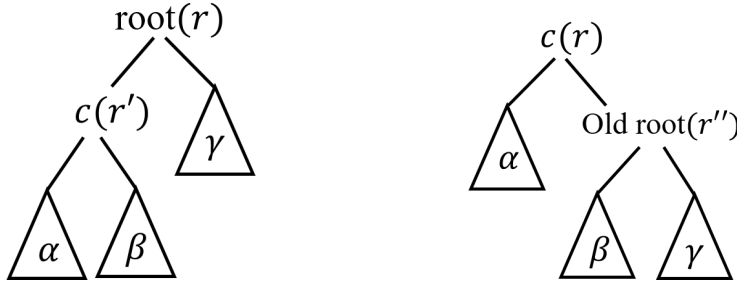


In the figure above, $\text{SPLAY}(x)$ takes four steps. Each r_i represents the rank at the corresponding node. If (1) and (2) are true, then

$$\begin{aligned} AC(\text{SPLAY}(X)) &= AC(\text{Zig-zag}) + AC(\text{Zig-zig}) + AC(\text{Zig-zag}) + AC(\text{Zig}) \\ &\leq 3(r_1 - r_0) + 3(r_2 - r_1) + 3(r_3 - r_2) + 3(r_4 - r_3) + 1 \\ &= 3(r_4 - r_0) - 1 \end{aligned}$$

which is the statement of Access Lemma.

We first show (1):



As labeled in the above figures, the rank of the root r stays constant throughout the rotation. Furthermore,

$$\Phi(\text{After Zig}) = r + r'' + \Phi(\alpha) + \Phi(\beta) + \Phi(\gamma)$$

$$\Phi(\text{Before Zig}) = r + r' + \Phi(\alpha) + \Phi(\beta) + \Phi(\gamma)$$

Hence

$$\begin{aligned} AC(\text{Zig}) &= 1 + \Delta\Phi \\ &= 1 + r'' - r' \\ &\leq 1 + r - r' \quad \text{since } r(P) \geq r(C) \\ &\leq 3(r - r') + 1 \end{aligned}$$

We now show (2): In the following portion of the proof, we let $r = r(GP)$, $b = r(P)$, $a = r(C)$ be the ranks *before* the rotations.

- Case 1: $r > a$

$$\Phi(\text{Before}) = r + b + a + \Phi(\text{subtrees}) \geq r + 2a + \Phi(\text{subtrees})$$

$$\Phi(\text{After}) \leq 3r + \Phi(\text{subtrees})$$

Hence,

$$\Delta\Phi \leq 3r - (r + 2a) = 2(r - a)$$

and

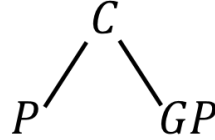
$$AC = 1 + \Delta\Phi \leq 1 + 2(r - a) \leq 3(r - a)$$

- Case 2: $r = a$

Since $r \geq b \geq a$, $r = a$ implies $r = b = a$. Hence

$$\Phi(\text{Before}) = r + b + a + \Phi(\text{subtrees}) = 3r + \Phi(\text{subtrees})$$

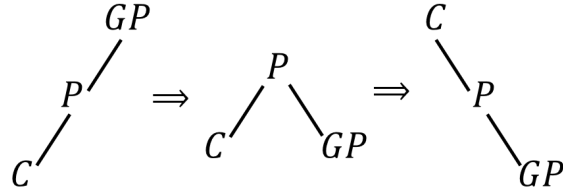
We claim that $\Phi(\text{After}) \leq 3r - 1 + \Phi(\text{subtrees})$. In the case of Zig-zag, the tree after the rotation would have the following structure with C as the new root:



By 4.11 we have $r(GP) + r(P) < 2(C)$. Therefore,

$$\Phi(\text{After}) = r(C) + r(GP) + r(P) + \Phi(\text{subtrees}) \leq 3r(C) - 1 + \Phi(\text{subtrees}) = 3r - 1 + \Phi(\text{subtrees})$$

In the case of Zig-Zig, we have



Note that from the left stage to the middle stage, the subtree rooted at C is the same. Hence $r(C) = r$ for both figures. In the middle stage, $r(P) = r(\text{root}) = r$. Hence $r(GP) < r$ by 4.10. And since the subtree rooted at GP is the same for the middle and the right stages, $r(GP) < r$ after the rotation. Therefore,

$$\Phi(\text{After}) = r(C) + r(P) + r(GP) + \Phi(\text{subtrees}) \leq r + r + (r - 1) + \Phi(\text{subtrees}) = 3r - 1 + \Phi(\text{subtrees})$$

Therefore for both Zig-zag and Zig-zig.

$$\Delta\Phi = \Phi(\text{after}) - \Phi(\text{before}) \leq -1$$

and

$$AC = 1 + \Delta\Phi \leq 1 + (-1) = 0 \leq 3(r - r)$$

□