Course: Comtemporary Algorithms T.II/2019-20

Lecture 2: Splay Tree and Treap

8 January 2020

Lecturer: Kanat Tangwongsan *Scribe: Pitipat Chairoj & Nuttapat Koonarangsri*

1 Splay Tree

In a Binary Search Tree, root is a great position. Operations such as insert, delete, and join will be easier if we promote a node to be the root. And, we can promote a node without violating BST rules.

Splay Tree is a self-balancing binary search tree. A node in a splay tree will get promoted to the root of the tree, by using rotations, every time the node is accessed.

However, promoting a node to be the root could be costly. Consider a binary search tree of n nodes labeled from 0 to $n-1$ with the node labeled $n-1$ as the root and the node labeled 0 being the leftmost node in the tree. With single rotations, if we promote 0, then 1, then 2, etc. sequentially, the total work is $\Omega(n^2)$ (with an amortized lower bound of $\Omega(n)$ per promotion). Splay tree do the rotation and restructuring in a very special way that guarantees a logarithmic amortized bound.

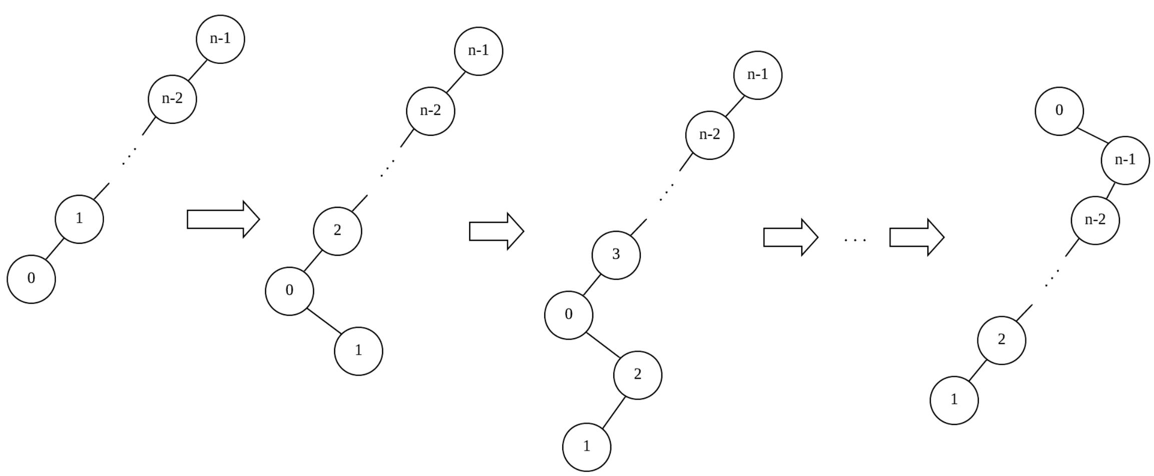
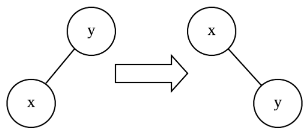


Figure 1: Promotion (with single rotations)

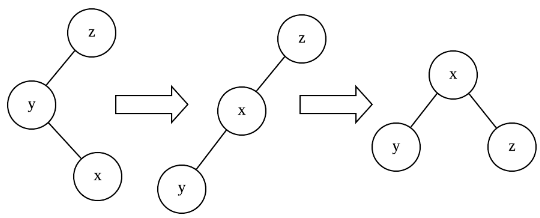
1.1 Rewrite Rules

In the following rules, x will be accessed, and we will apply rules until x becomes the Hello root



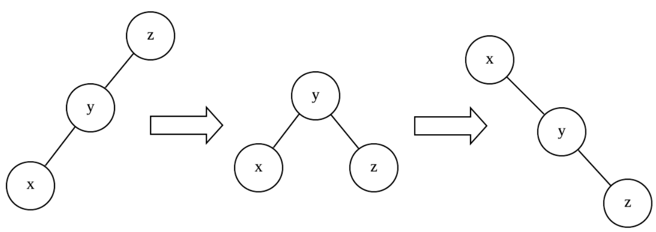
1.

Zig (single rotation)



2.

Zig-Zag (double rotations)



3.

Zig-Zig (double rotations)

Note that each rule has a symmetric version. **splay(x)** refers to applying rewrite rules to promote a node x to be the root.

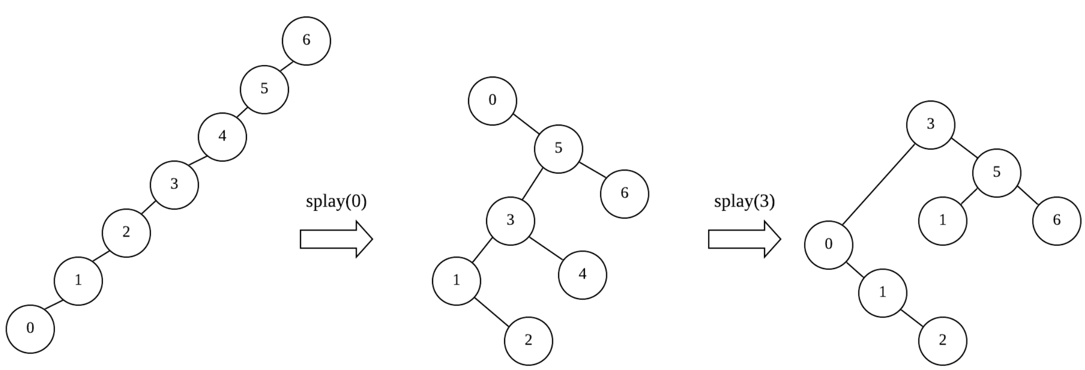


Figure 2: Splaying Example

1.2 The Amortized Analysis

Recap: Potential Method

$$\text{Amortized cost} = \frac{\text{total cost of a sequence of operations}}{\text{total number of operations}}$$

$$\underbrace{\Phi(D)}_{\text{potential function}} = \text{reserve stored in data structure } D \text{ at the point}$$

Let say $\sigma_1, \sigma_2, \ldots, \sigma_m$ are operations performed on data structure D, and the cost of operation $\sigma_i = c_i$ Initially, D is at state s_0, and σ_i changes state $s_{i-1} \to s_i$. Then, the amortized cost of operation σ_i is given by

$$A_i = c_i + \Phi(s_i) - \Phi(s_{i-1})$$

via summation:

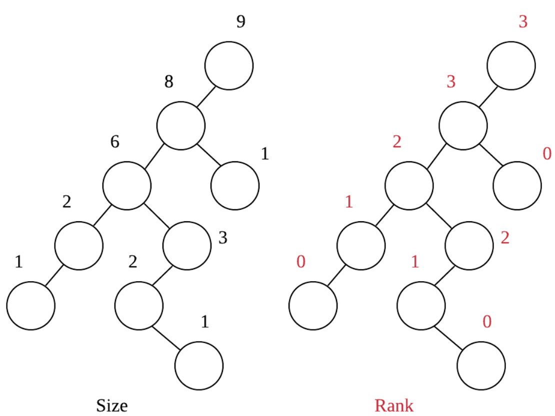
$$\sum A_i = \sum c_i + \Phi(s_m) - \Phi(s_0) \tag{1}$$

Steps:

Step 1: Choose a potential function.

Step 2: Prove that amortized cost satisfied the bound.

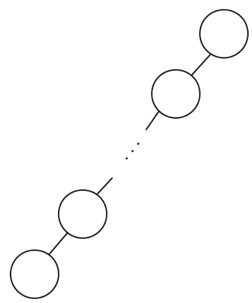
Step 3: Bound $\Phi(s_m) - \Phi(s_0)$ appropriately.



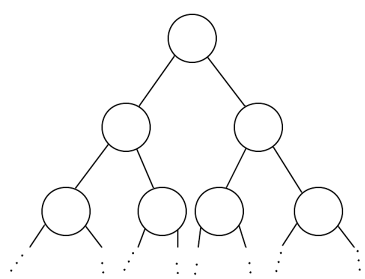
Take any tree T. For a node $x \in T$, define

$$\begin{aligned} s(x) &:= \text{number of nodes inside the subtree rooted at x} \\ r(x) &:= \lfloor \log_2(s(x)) \rfloor \\ \Phi(T) &:= \sum_{x \in T} r(x) = \sum_{x \in T} \lfloor \log_2(s(x)) \rfloor \end{aligned}$$

Example

- 

  Lopsided tree: $\Phi \approx \sum_{k=1}^{n} \log_2(k) \in \Theta(n\log(n))$

- 

  Perfect Binary Search tree: $\Phi \approx \sum_{i=0}^{\log_2(n)} 2^i(\log_2(n) - i) \in \Theta(n)$

Claim 1.1. *Suppose p is the root of a subtree with rank $r(p)$, a and b are children of p and are sibling of each other, then $r(a) \leq r(p) - 1$ or $r(b) \leq r(p) - 1$.*

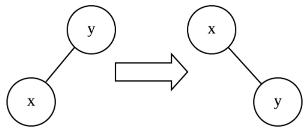
Lemma 1.2 (Access Lemma)**.** *In a tree T with root t, splaying x yields a new tree T' such that*

$$\textit{the amortized cost} \leq 3(r(t) - r(x)) + 1$$

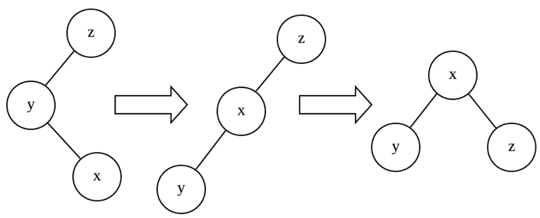
.

Proof. (Sketch)
Let y be parent of x and z be grandparent of x where x is a node in a tree T.

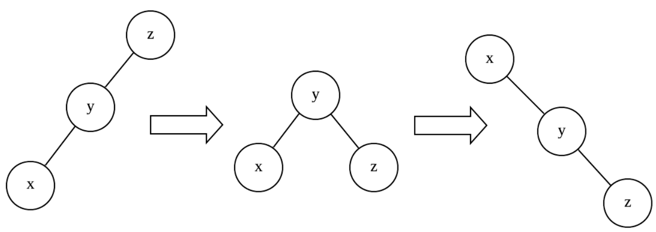
Zig Case Happens at most once at the end (as x becomes the root). Thus, amortized cost $\leq 3(r(y) - r(x)) + 1$.



Zig-Zag Case amortized cost $= r'(x) - r(x) + r'(y) - r(y) + r'(z) - r(z) + 1$. Since, $r'(x) = r(z)$, amortized cost $= r'(z) - r(x) + r'(y) - r(y) \leq 3(r'(x) - r(x))$.



Zig-Zig Case amortized cost $\leq 3(r'(x) - r(x))$.



Sum them up, they telescope. $\square$

Theorem 1.3. *Any sequence of m operations in a tree with $\leq n$ nodes costs $\leq O(mlog(n) + nlog(n))$.*

Proof. Let T_0 be tree's initial state, T_m be tree's state after m operations, and

$$\Phi(T) = \sum_{x \in T} r(x) = \sum_{x \in T} \lfloor log_2(s(x)) \rfloor$$

To find the total cost, we can rearrange (1) to be

$$\sum c_i = \sum A_i + \Phi(T_0) - \Phi(T_m)$$

From Access Lemma, we know that amortized cost A_i of every operation is bounded by $3(\underbrace{r(t_i) - r(x_i)}_{\leq \log(n)})+$

1 where t_i is the root and x_i is the node we are splaying at operation σ_i. Hence,

$$\begin{aligned}
\sum c_i &= \sum A_i + \Phi(T_0) - \Phi(T_m) \\
&\leq \sum A_i + \Phi(T_0) \\
&= m(3\text{log}(n) + 1) + n\text{log}(n) \\
&= 3m(\text{log}(n) + m + n\text{log}(n) \\
&= O(m\text{log}(n) + n\text{log}(n))
\end{aligned}$$

□

1.3 Operations

Searching: same as BST, so $O(\text{log}(n))$

Insertion: steps of inserting a node x

Step 1: Look up x and find where the node x is going to attach to. Say that node is p.

Step 2: Promote p to be the root.

Step 3: Insert x to left/right of p depending on whether $x < p$ or $x > p$.

amortized cost of insertion is $O(\text{log}(n))$

Join: steps of joining two BSTs A and B

Step 1: Promote leftmost node a of A to be the root.

Step 2: Make B the right child of a.

amortized cost of join is $O(\text{log}(n))$

Deletion: steps of deleting a node x

Step 1: Promote x to be the root.

Step 2: Remove x.

Step 3: Join the remaining subtree.

amortized cost of deletion is $O(\text{log}(n))$

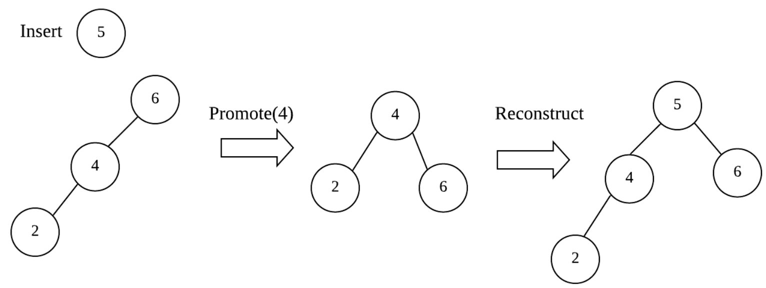


Figure 3: $x > p$

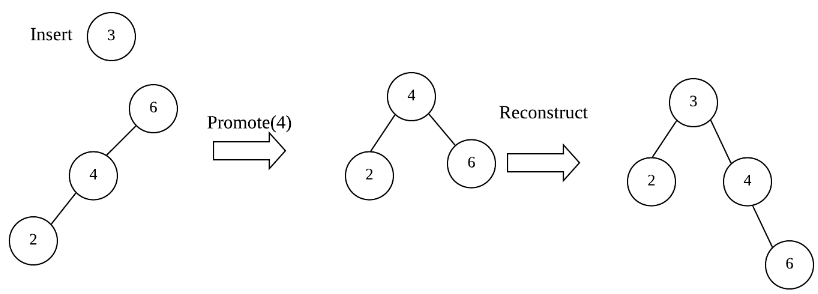


Figure 4: $x < p$

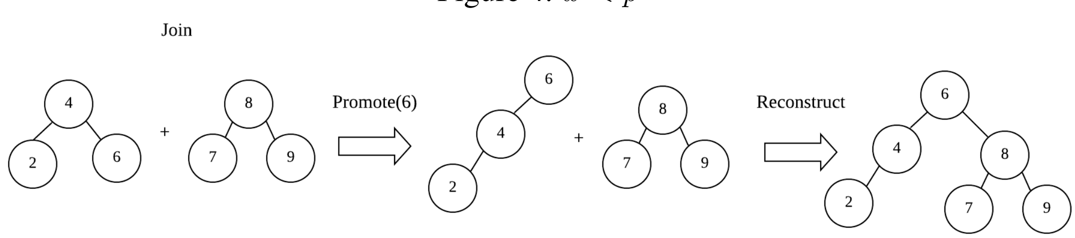


Figure 5: Join two BSTs

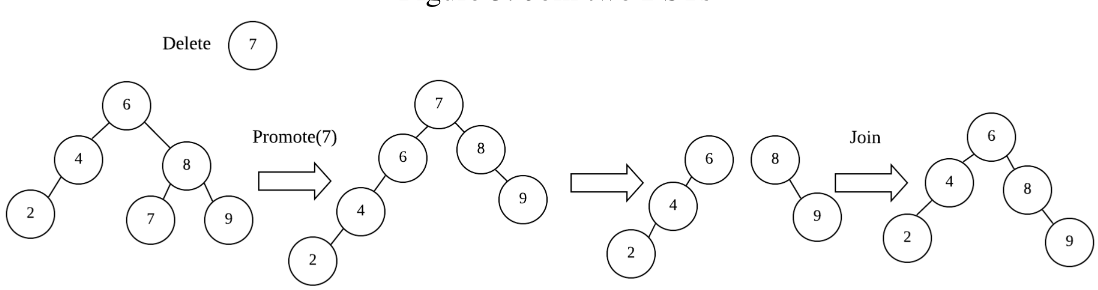


Figure 6: Delete a node

2 Treap

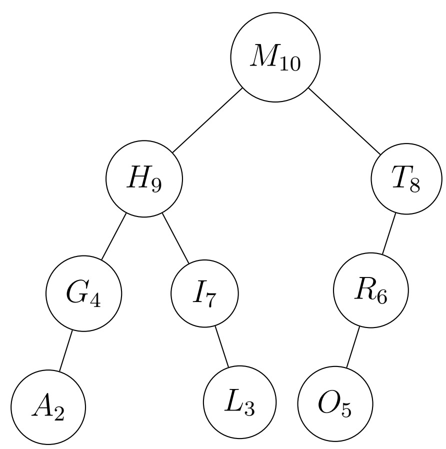
A Treap is a combination of the Binary Search Tree and the Binary Heap. The node of this tree store a pair of (X,Y) where X represents binary search key and Y represents a binary heap. There are many heap variant for Treap but in this lecture we will focus on the Max heap.

Assuming that all X and all Y are different, if a node N contains value (X_0, Y_0) then all nodes in the left subtree have $X < X_0$ and the right one have $X > X_0$, as a normal binary search tree. Moreover, all nodes in both left and right subtree will have a priority value, Y, with $Y < Y_0$

Consider the following list of Xs and the corresponding priorities Ys.

$$\begin{bmatrix} H & M & T & G & I & A & L & O & R \\ 9 & 10 & 8 & 4 & 7 & 2 & 3 & 5 & 6 \end{bmatrix}$$

Such pairs will correspond to the following Treap :



(2)

Node Properties The node in a Treap must satisty these two properties:

1. **BST Property** The keys are stored in-order in the tree

2. **Heap Property** The priorities satisfy the heap property (max-heap). The Max-heap property requires that the value of every node to the left must be less than the node, and the nodes to the right must be greater.

2.1 Assumption

. In this version of Treap, we will assume that all key and priorities are unique. Moreover, the priorities are random.

Theorem 2.1. *There is a unique Treap for a set of keys and priorities.*

2.2 Operations

- **Insert(x,y)** in O(depth(x))
- **Search(x)** in O(depth(x))
- **Delete(x)** in O(depth(x)), equivalent to inserting x backward.

2.3 How deep is a treap ?

$\mathbb{E}$[depth(X)] = $\boldsymbol{O}(\log n)$

Claim 2.2. *depth* $\leq \boldsymbol{O}(\log n)$ *whp.*

Lemma 2.3 (Ancestor Lemma)**.** *Let* $x_1 < x_2 < \cdots < x_n$ *be the search key of a Treap with corresponding priorities* p_i*. The lemma state that* x_j *is ancestor of* x_i *if and only if* x_j *has the highest priority among the keys between* x_i *and* x_j*.*

Lemma 2.4. *Because priorities are chosen at random thus*

$$\mathbf{Pr}[x_j \text{is ancestor of } x_i] = \frac{1}{|i-j|+1}$$

Proof. Let $A_{i,j} = 1_{[x_i \text{is ancestor of} x_j]}$

$$\begin{aligned}
depth(x_i) &= \sum_j A_{i,j} \\
\mathbb{E}[depth(x_i)] &= \sum_j \mathbb{E}[A_{i,j}] \\
&= \sum_{j=1}^{i} \frac{1}{|i-j|+1} + \sum_{j=i+1}^{n} \frac{1}{|j-i|+1} \\
&= H_i + H_{n-i+1} \\
&\leq 2\ln n + 2
\end{aligned}$$

where H_i is harmonic number $H_i = \sum_i^n \frac{1}{n}$ with the bound $\ln n < H_n < 1 + \ln n$
$\therefore$ depth(x) $\leq \boldsymbol{O}(\log n)$

□

What about high probability ? For a fixed i, $A'_{i,j}s$ are independent. Consider $depth(x_i)$

Theorem 2.5. *Chernoff-Hoeffding Let* $X = x_1 + x_2 + \cdots + x_n$ *where* x_i*'s are independently distributed in* $[0, 1]$*. Then for* $\lambda > 0$*;*

$$\mathbf{Pr}[X > (1+\lambda)\mathbb{E}[x]] \leq \exp -\frac{\lambda^2}{3}\mathbb{E}[x]$$

$$depth(x_i) = \sum_{j=1}^{i-1} A_{ij} + \sum_{j=i+1}^{n} A_{ij} + 1 \quad \textbf{Left, Right, and itself respectively}$$

we want to bound $\mathbf{Pr}[\textbf{depth} \leq 8 \ln n] \geq 1 - \dots \frac{1}{n^0}$

$$\begin{aligned} \mathbf{Pr}[\textbf{depth} \leq 8 \ln n] \geq 1 - \dots \frac{1}{n^0} \\ \leq \mathbf{Pr}[\textbf{Left} > 4 \ln n] + \mathbf{Pr}[\textbf{Right} > 4 \ln n] \end{aligned}$$

but $\mathbf{Pr}[\textbf{depth} \leq 8 \ln n] = 1 - \mathbf{Pr}[\textbf{depth} > 8 \ln n]$, and since $\mathbf{Pr}[R > 4 \ln n]$ is symmetric to the left, we can just focusing on one side.

$$\begin{aligned} \mathbb{E}[Left] &= H_i - 1 \\ &\leq \ln n + 1 - 1 \\ &= \ln n \\ \mathbf{Pr}[Left > 4 \ln n] &= \mathbf{Pr}[(1+3) \ln n] \quad \text{Notice we have the form } \mathbf{Pr}[(1+\lambda) \ln n] \\ &\leq \exp\left(-\frac{3^2}{3} \mathbb{E}[Left]\right) \\ &\leq \frac{1}{n^3} \\ \therefore \mathbf{Pr}[\text{depth} \leq 8 \ln n] &\geq 1 - \frac{2}{n^3} \quad \blacksquare \end{aligned}$$

2.4 Relation with QuickSort

qs(x):

- Randomly pick a pivot p
- Split into $< p, = p, > p$
- Recurse step 1

Notice that the pivot diagram is look just like a Treap.