Analysis of the 'Search' operation

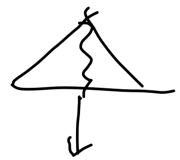
Let *T* be a treap with elements $x_1, \ldots, x_n$ x_i has *i*-th smallest key

n-th Harmonic number: $$H_n = \sum_{k=1}^{n} 1/k$$

Lemma:

1. Successful search: The expected number of nodes on the path to x_m is $H_m + H_{n-m+1} - 1$. $O(\log n)$

2. Unsuccessful search : Let *m* be the number of keys that are smaller than the search key *k*. The expected number of nodes on the search path is $H_m + H_{n-m}$. $O(\log n)$



Analysis of the 'Search' operation

Proof: Part 1

$$X_{m,i} = \begin{cases} 1 & x_i \text{ is ancestor of } x_m \\ 0 & \text{otherwise} \end{cases}$$

X_m = # nodes on the path from the root to x_m (incl. x_m)

$$X_m = 1 + \sum_{i<m} X_{m,i} + \sum_{i>m} X_{m,i}$$

$$E[X_m] = 1 + E\left[\sum_{i<m} X_{m,i}\right] + E\left[\sum_{i>m} X_{m,i}\right]$$

Analysis of the 'Search' operation

$i < m$:

$$E[X_{m,i}] = \text{Prob}[x_i \text{ is ancestor of } x_m] = 1/(m - i + 1)$$

All elements in $\{x_i, \ldots, x_m\}$ have the same probability of being the one with the smallest priority.

$\text{Prob}[P_{min}(\{x_i,\ldots,x_m\}) = x_i] = 1/(m-i+1)$

$i > m$:

$$E[X_{m,i}] = 1/(i - m + 1)$$

Analysis of the 'Search' operation

$$E[X_m] = 1 + \sum_{i<m} \frac{1}{m-i+1} + \sum_{i>m} \frac{1}{i-m+1}$$

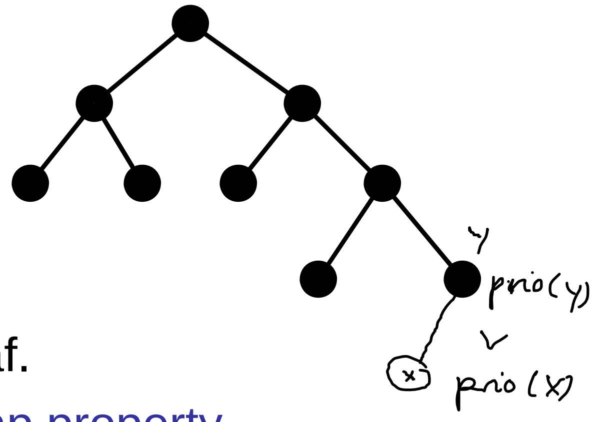
$$= 1 + \frac{1}{m} + \ldots + \frac{1}{2} + \frac{1}{2} + \ldots + \frac{1}{n-m+1}$$

$$= H_m + H_{n-m+1} - 1$$

Part 2 follows analogously

Inserting a new element *x*

1. Choose prio(x).   uniformly at random in [0,1)
2. Search for the position of *x* in the tree.



- We maintain the search tree property
- We may violate the heap property
- We must restore the heap property

3. Insert *x* as a leaf.
4. Restore the heap property.

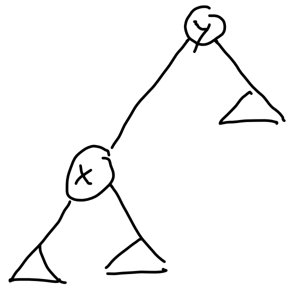
   while prio(parent(*x*)) > prio(*x*) **do**

   if *x* is left child **then** RotateRight(parent(*x*))

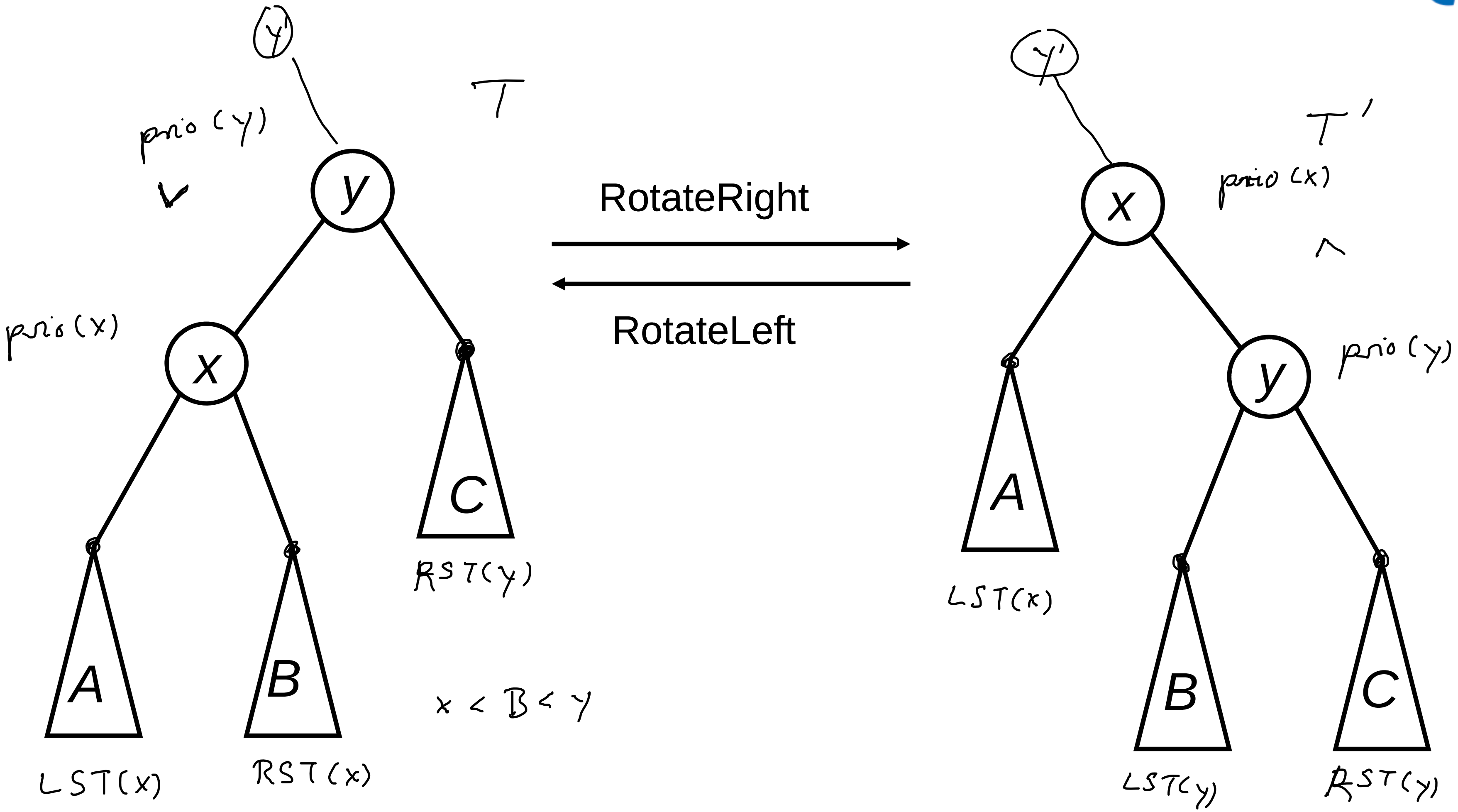
   else RotateLeft(parent(*x*));

   endif

   endwhile;



Rotations

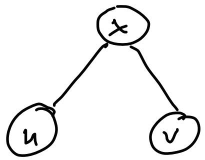


The rotations maintain the search tree property and restore the heap property.

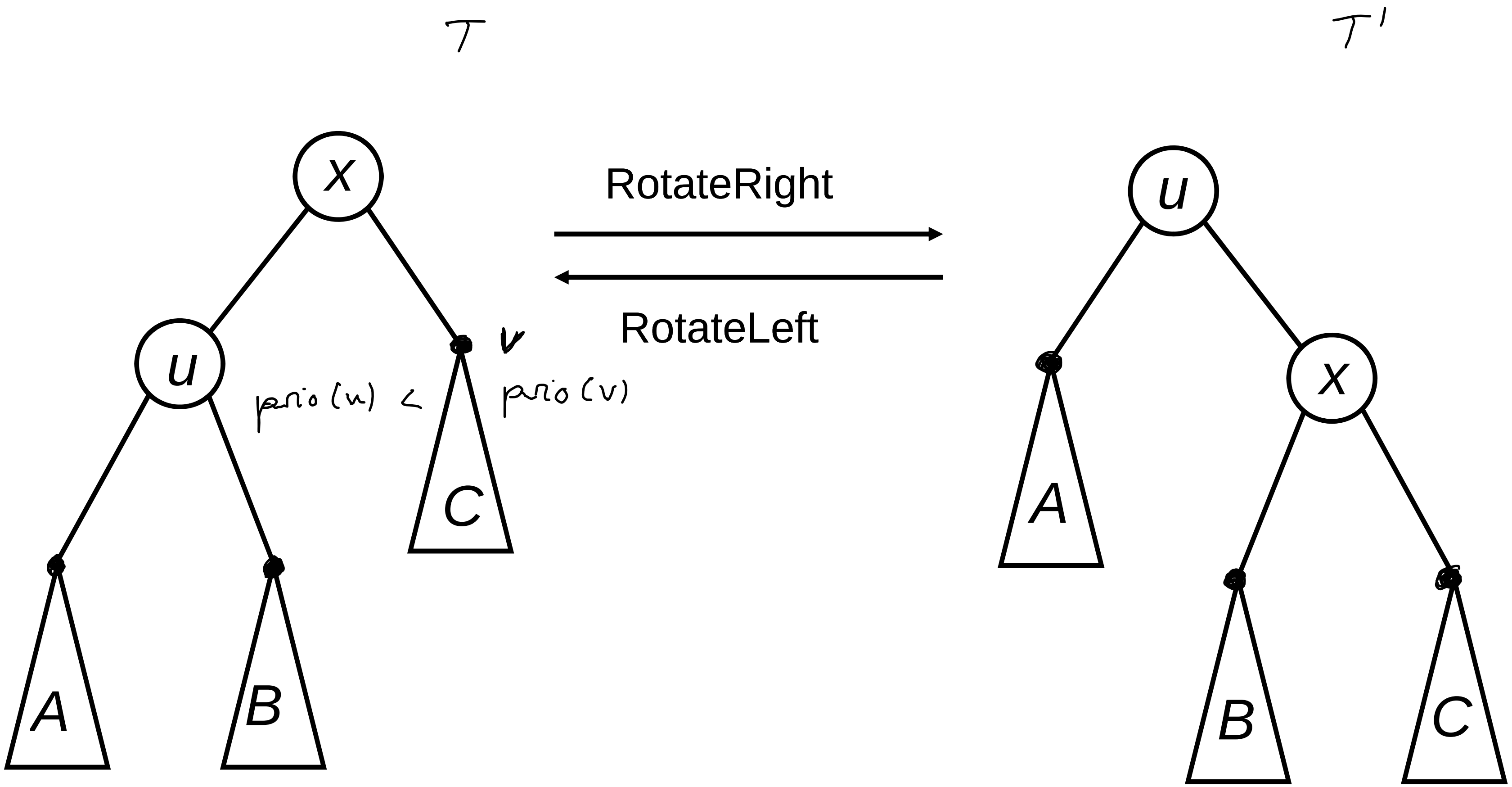
Deleting an element *x*

1. Find *x* in the tree.
2. **while** *x* is not a leaf **do**
   u := child with smaller priority;
   if *u* is left child **then** RotateRight(*x*))
   else RotateLeft(*x*);
   endif;
   endwhile;
3. Delete x;

- Deleting a leaf is easy
- Rotate an element to be deleted down in the treap until it becomes a leaf



Rotations



Analysis of 'Insert' and 'Delete' operations

Lemma: The expected running time of insert and delete operations is O(log *n*). The expected number of rotations is less than 2.

Proof: Analysis of insert (delete is the inverse operation)

\# rotations = depth of x after being inserted as a leaf (1)

- depth of x after the rotations (2)

Let $x = x_m$.

(2) Expected depth is $H_m + H_{n-m+1} - 1$. "successful search"

(1) Expected depth is $H_{m-1} + H_{n-m} + 1$. "unsuccessful search"

Before we insert element x_m: The tree contains *n-1* elements, *m-1* of them being smaller.

≤ 0 ≤ 0

$$\mathbb{E}[\#\text{ rotations}] = \underbrace{H_{m-1} + H_{n-m} + 1}_{(1)} - (\underbrace{H_m + H_{n-m+1} - 1}_{(2)}) < 2$$

Extended set of operations

n = number of elements in treap T.

- Minimum(T): Return the smallest key. O(log n)
- Maximum(T): Return the largest key. O(log n)
- List(T): Output elements of S in increasing order. O(n)

  inorder traversal

  List (T) = { List (LST(T)); output (root (T)); List (RST (T))}

- Union(T_1,T_2): Merge T_1 and T_2.

  Condition: $\forall x_1 \in T_1, x_2 \in T_2$: key($x_1$) < key($x_2$)

- Split(T,k,T_1,T_2): Split T into T_1 and T_2.

  $\forall x_1 \in T_1, x_2 \in T_2$: key($x_1$) $\leq k$ and $k <$ key(x_2)

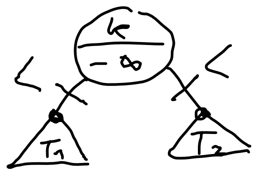
The 'Split' operation

Split(T,k,T_1,T_2): Split T into T_1 and T_2.

$\forall\, x_1 \in T_1,\ x_2 \in T_2$: key($x_1$) $\le k$ and key(x_2) $> k$

W.l.o.g. key k is not in T.

Otherwise delete the element with key k and re-insert it into T_1 after the split operation.



1. Generate a new element x with key(x)=k and prio(x) = -∞.
2. Insert x into T.
3. Delete the new root. The left subtree is T_1, the right subtree is T_2.

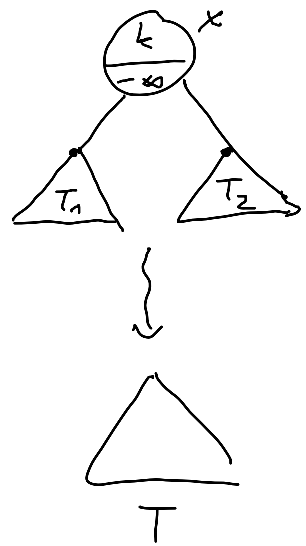
$x_1 \in T_1$: key$(x_1) \le k$ $x_2 \in T_2$: key$(x_2) > k$

The ‘Union’ operation

Union(T_1,T_2): Merge T_1 and T_2.

Condition: $\forall\, x_1 \in T_1,\, x_2 \in T_2$: key($x_1$) < key($x_2$)

1. Determine key k with key(x_1) < k < key(x_2) for all $x_1 \in T_1$ and $x_2 \in T_2$.
2. Generate element x with key(x)=k and prio(x) = -∞.
3. Generate treap T with root x, left subtree T_1 and right subtree T_2.
4. Delete x from T.



Analysis

Lemma: The expected running time of the operations Union and Split is $O(\log n)$.

Implementation

Priorities from [0,1)

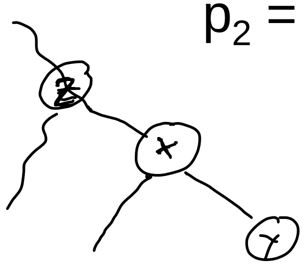
Priorities are used only when two elements are compared to find out which of them has the higher priority. insert.

In case of equality, extend both priorities by bits chosen uniformly at random until two corresponding bits differ.

$p' = 0.0101111$

$p_1 = 0.010111001$ 0 0 0 0

$p_2 = 0.010111001$



$p_1 = 0.010111001011$

$p_2 = 0.010111001010$