Algorithms Theory

04 - Treaps

Dr. Alexander Souza

The dictionary problem

Given: Universe $(U,<)$ of keys with a total order

Goal: Maintain set $S \subseteq U$ under the following operations

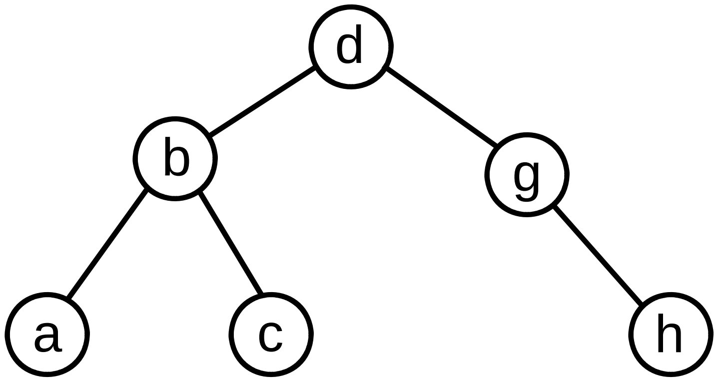
- Search(x,S): Is $x \in S$?
- Insert(x,S): Insert x into S if not already in S.
- Delete(x,S): Delete x from S.

Extended set of operations

- Minimum(S): Return smallest key.
- Maximum(S): Return largest key.
- List(S): Output elements of S in increasing order of key.
- Union(S_1,S_2): Merge S_1 and S_2 .
  Condition: $\forall x_1 \in S_1, x_2 \in S_2: \; x_1 < x_2$
- Split(S,x,S_1,S_2): Split S into S_1 and S_2.
  $\forall x_1 \in S_1, x_2 \in S_2: \; x_1 \leq x$ and $x < x_2$

Known solutions

- **Binary search trees**



Drawback: Sequence of insertions may lead to
a linear list a, b, c, d, e, f

- **Height balanced trees:** AVL trees, (a,b)-trees

Drawback: Complex algorithms or high memory requirements.

Approach for randomized search trees

If n elements are inserted in random order into a binary search tree, the expected depth is 1.39 log n.

Idea: Each element x is assigned a priority chosen uniformly at random

$$\text{prio}(x) \in R$$

The goal is to establish the following property.

(*) The search tree has the structure that would result if elements were inserted in the order of their priorities.

Treaps (Tree + Heap)

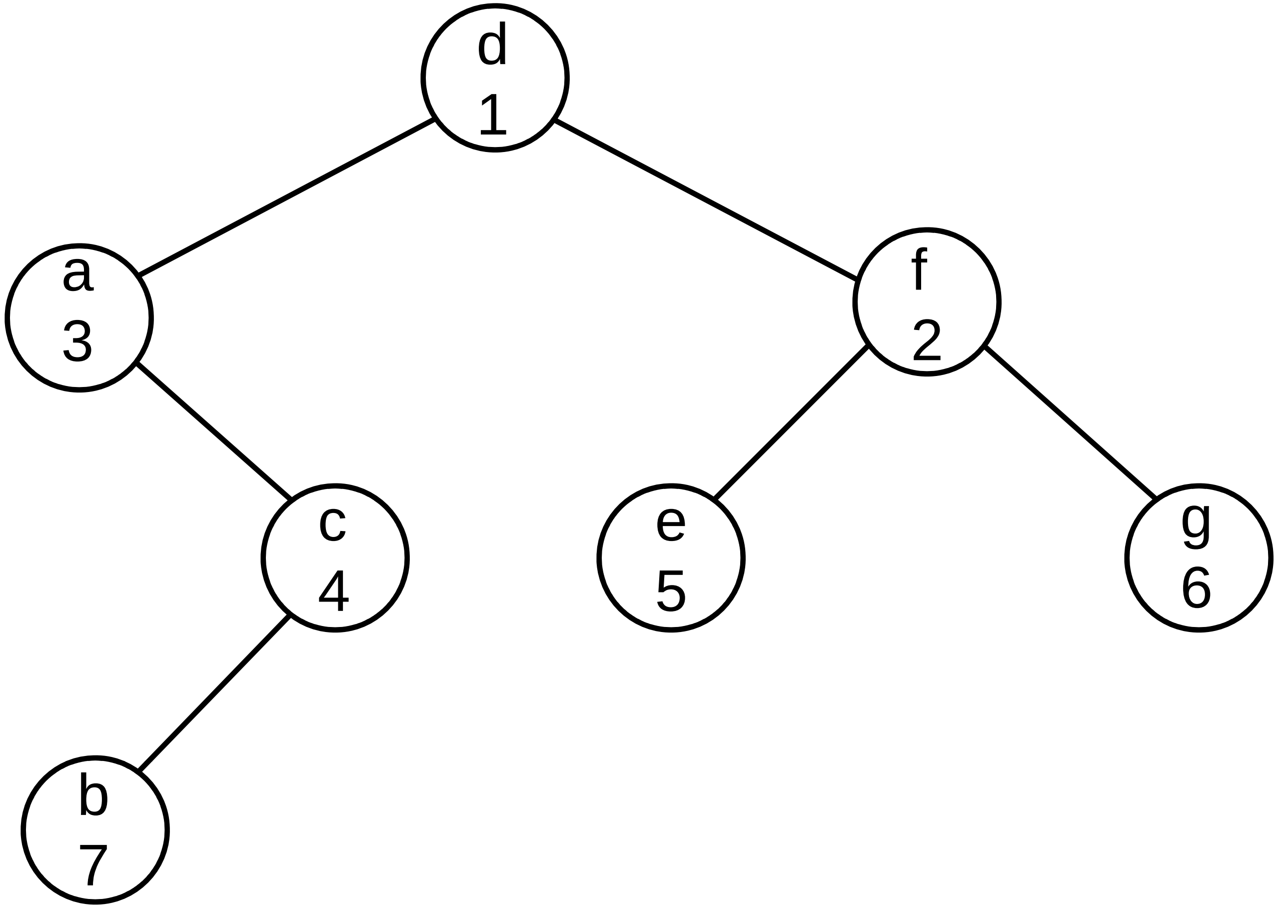
Definition: A treap is a binary tree.
Each node contains one element x with key(x) $\in U$ and prio(x) $\in R$.
The following properties hold.

- Search tree property
  For each element x:
  - elements y in the left subtree of x satisfy: key(y) < key(x)
  - elements y in the right subtree of x satisfy : key(y) > key(x)

- Heap property
  For all elements x,y:
  If y is a child of x, then prio(y) > prio(x).
  All priorities are pairwise distinct.

Example

key	a	b	c	d	e	f	g
priority	3	7	4	1	5	2	6



Treap uniqueness

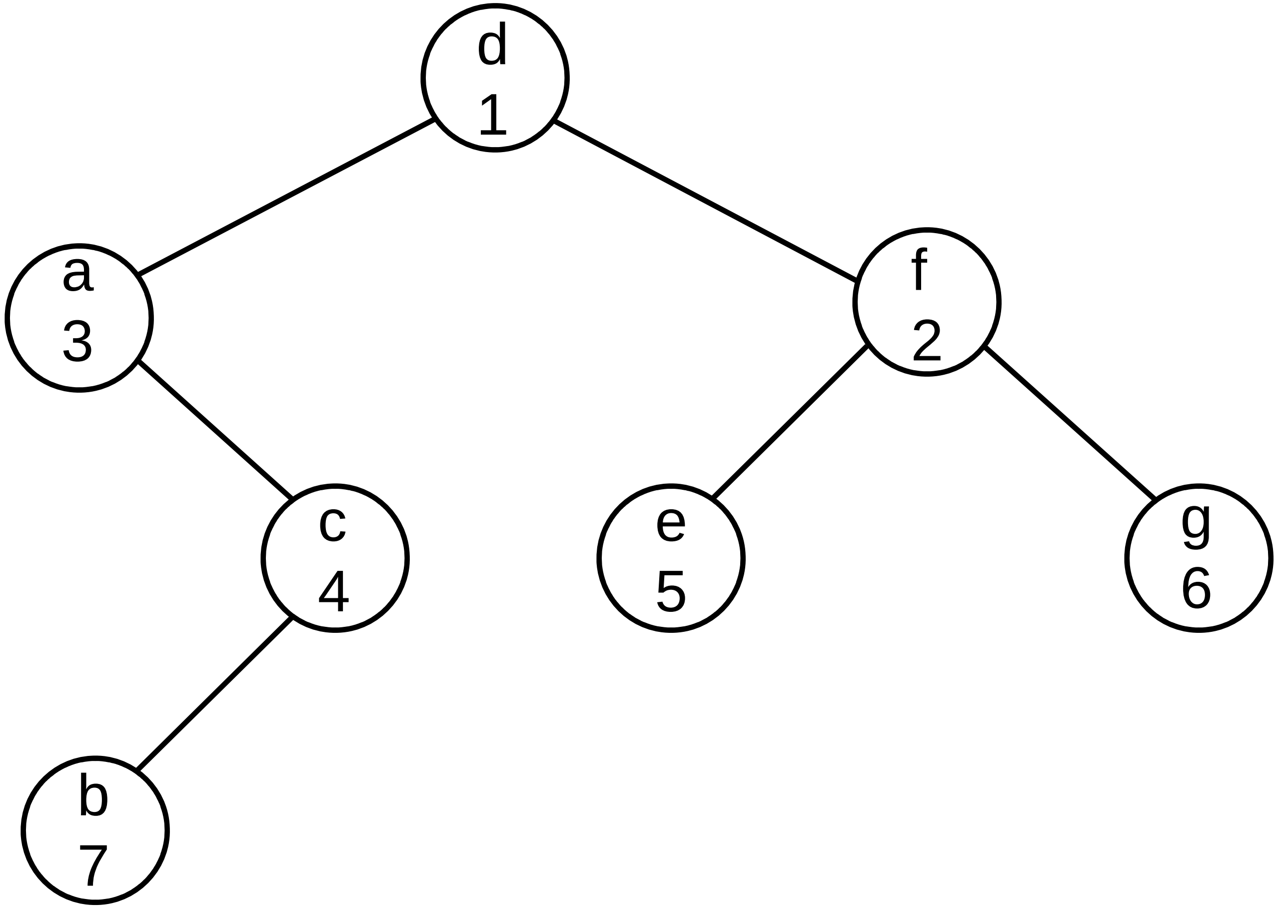
Lemma: For elements $x_1, \ldots, x_n$ with key(x_i) and prio(x_i), there exists a unique treap. It satisfies property (*).

Proof:

n=1: ok

n>1:

Search for an element



Search for element with key k

```
v := root;
while v ≠ nil do
   case key(v) =  k : stop;   "element found" (successful search)
             key(v) <  k : v:= RightChild(v);
             key(v) >  k : v:= LeftChild(v);
   endcase;
endwhile;
"element not found" (unsuccessful search)
```

Running time: O(# elements on the search path)

Analysis of the search path

Elements $x_1, \ldots, x_n$ $\qquad$ x_i has i-th smallest key

Let M be a subset of the elements.

$P_{min}(M)$ = element in M with lowest priority

Lemma:

a) Let $i<m$. $\quad x_i$ is ancestor of x_m iff $P_{min}(\{x_i,\ldots,x_m\}) = x_i$

b) Let m<i. $\quad x_i$ is ancestor of x_m iff $P_{min}(\{x_m,\ldots,x_i\}) = x_i$

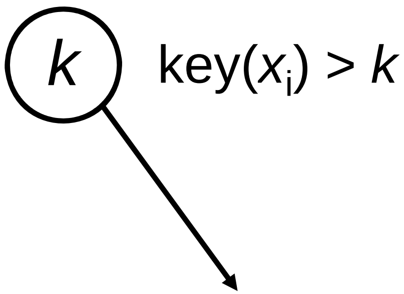
Analysis of the search path

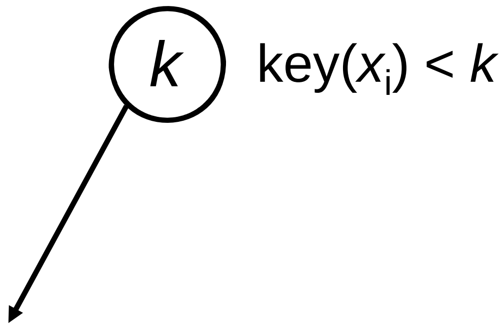
Proof: a) Use (*). Elements are inserted in order of increasing priorities.

"$\Leftarrow$" $P_{min}(\{x_i,\ldots,x_m\}) = x_i \quad \Rightarrow \quad x_i$ is inserted first among $\{x_i,\ldots,x_m\}$.

When x_i is inserted, the tree contains only keys k with

$k < \text{key}(x_i)$ or $k > \text{key}(x_m)$





Analysis of the search path

Proof: a) (Let $i<m$. x_i is ancestor of x_m iff $P_{min}(\{x_i,\dots,x_m\}) = x_i$)
"$\Rightarrow$" Let $x_j = P_{min}(\{x_i,\dots,x_m\})$. Show: $x_i = x_j$

Suppose: $x_i \neq x_j$

Case 1: $x_j = x_m$

Case 2: $x_j \neq x_m$

Part b) follows analogously.

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

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 $H_m + H_{n-m}$.

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

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

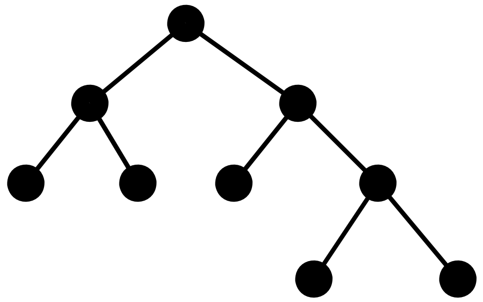
$$= 1 + \frac{1}{m} + \ldots + \frac{1}{2} + \frac{1}{2} + \ldots + \frac{1}{n-m+1}$$

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

Part 2 follows analogously

Inserting a new element *x*

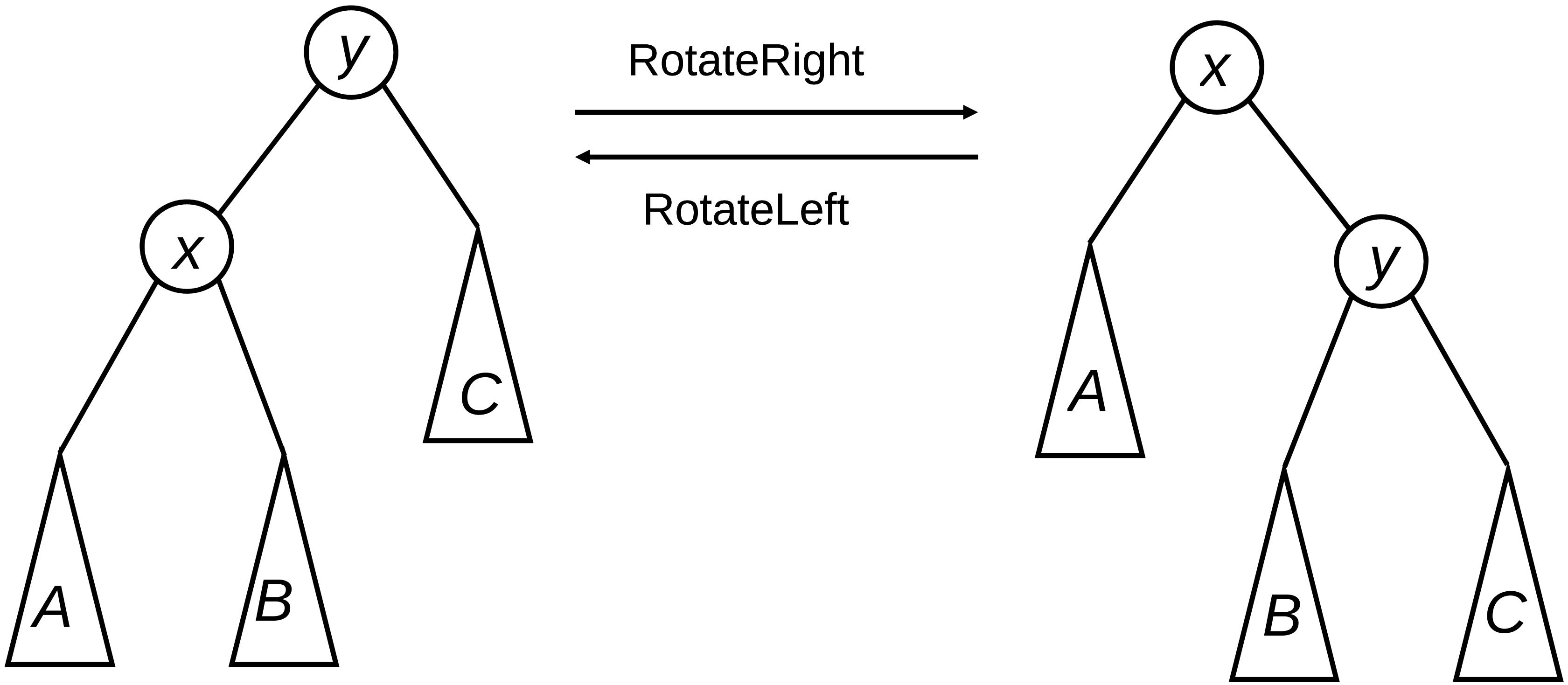
1. Choose prio(x).
2. Search for the position of *x* in the tree.



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*))

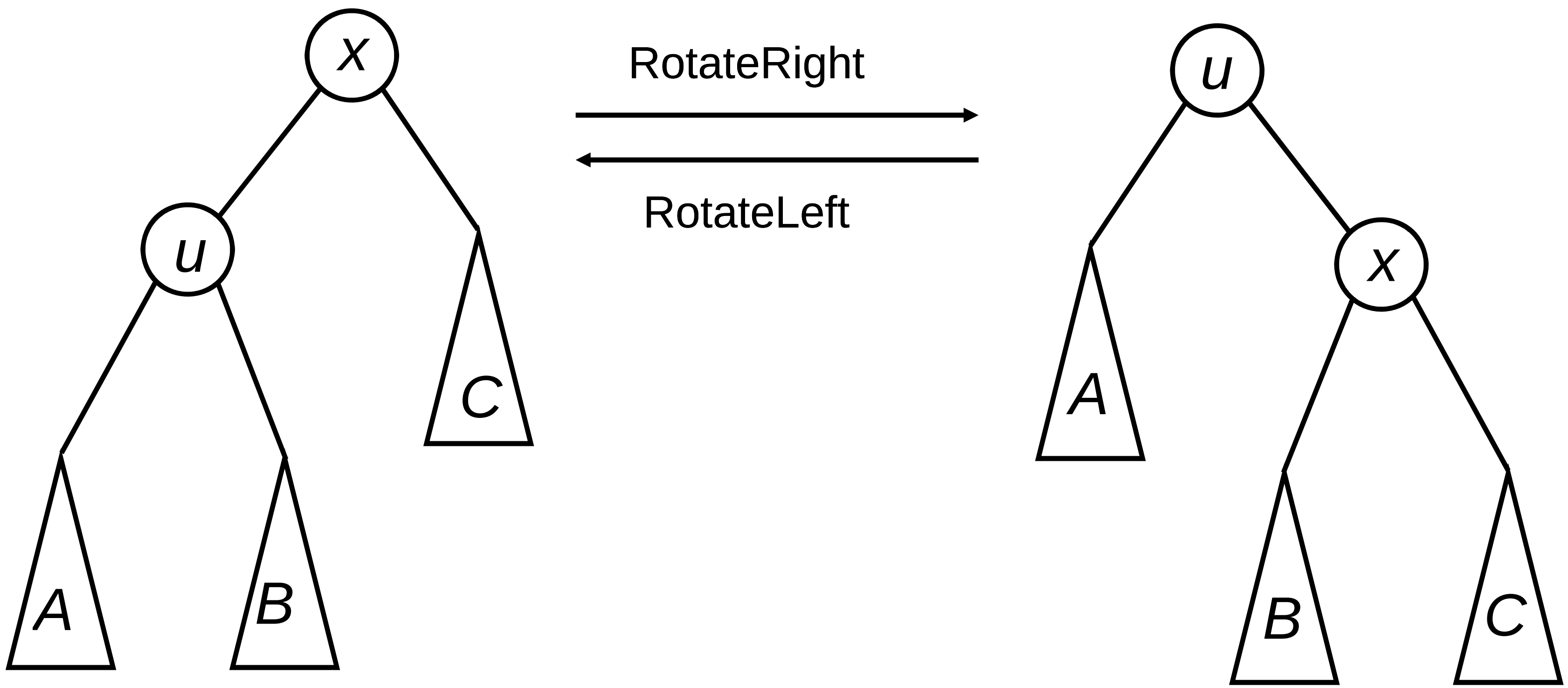
    else RotateLeft(*x*);

    endif;

    endwhile;

3. Delete x;

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

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

The tree contains $n-1$ elements, $m-1$ of them being smaller.

rotations = $H_{m-1} + H_{n-m} + 1 - (H_m + H_{n-m+1} - 1) < 2$

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)

- 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($x_1$) $\leq k$ 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) = $-\infty$.
2. Insert x into T.
3. Delete the new root. The left subtree is T_1, the right subtree is T_2.

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) = $-\infty$.
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.

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

$p_1 = 0.010111001$ $\qquad$ $p_1 = 0.010111001\textcolor{red}{011}$

$p_2 = 0.010111001$ $\qquad$ $p_2 = 0.010111001\textcolor{red}{010}$